Paper reproduction project Segnet - Group 26

Hao Li 5008891, Zixuan Wan 5059747, Noureddine Begga 4384261

April 20, 2020

0.1 Our take on the Impact of Illumination-Invariant Image Pre-transformation for Contemporary Automotive Semantic Scene Understanding.

In 2018, Naif Alshammari, Samet Akcay and Toby P. Breckon published their paper On the Impact of Illumination-Invariant Image Pre-transformation for Contemporary Automotive Semantic Scene Understanding. The authors of this paper realized that illumination changes in outdoor environments under non-ideal weather conditions have a negative impact on automotive scene understanding and segmentation performance. The paper presents an evaluation of illumination-invariant image transforms on the CamVid dataset, to see if this improves the state of the art in scene understanding performance.

As part of TU Delft's CS4240 Deep Learning course, we — Noureddine Begga, Hao Li and Zixuan Wan— attempt to give the reproduction of the results achieved in said paper a try. That is, we try to develop an implementation of the Deep Fully Convolutional Neural Network SegNet and preprocess the CamVid dataset with the illumination invariant image representation. More specifically, our goal is to achieve similar results as Table 1 of the paper (which is copied below).

Method	Sky	Building	Pole	Road	Pavement	Tree	SignSymbol	Fence	Car	Pedestrian	Bicyclist	Class avg.	Global acc.	mIoU	Precision	Recall
Original (RGB) [1]	0.73	0.846	0.33	0.87	**0.91**	0.76	0.43	0.41	0.73	0.60	0.11	0.61	0.807	0.46	**0.70**	0.61
$\mathcal{I}_{\text{Álvarez}}$ [7]	0.67	0.73	0.22	0.69	0.67	0.75	0.35	0.28	0.63	0.26	0.017	0.46	0.68	0.33	0.46	0.48
$\mathcal{I}_{\text{Maddern}}$ [6]	0.92	0.80	0.20	0.932	0.53	0.62	0.38	0.17	0.61	0.51	0.07	0.54	0.78	0.40	0.64	0.65

TABLE 1: Quantitative results are shown as accuracy of the CNN SegNet approach on CamVid test data for RGB and two methods.

The Cambridge-driving Labeled Video Database (CamVid) is the first collection of videos with object class semantic labels, complete with metadata. The database provides ground truth labels that associate each pixel with one of 32 semantic classes.

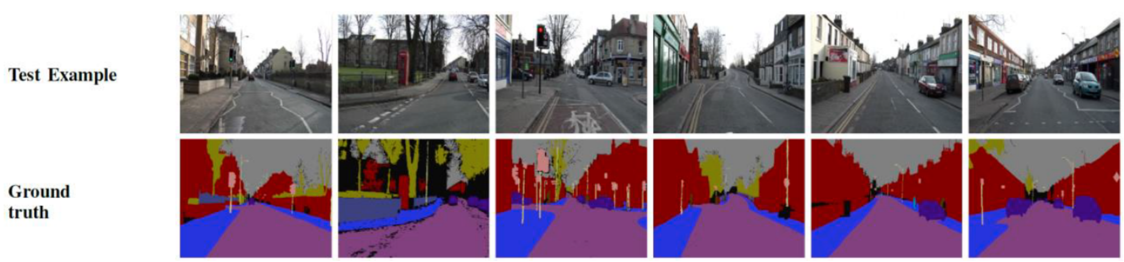


Figure 1: Obtained RGB images and their respective ground truth class labels

0.2 What is a illumination invariant image representation?

An illumination invariant image representation is a colour representation computed from RGB that removes (or minimises) scene colour variations due to varying scene lighting conditions. This technique was introduced as an intrinsic image to represent the illumination invariant and intrinsic properties in the image [1] with illumination transforms generally computed with reference to the physical properties behind the capture and the presence of colour within the space. In most literature where a type of illumination invariance is applied had as an objective to remove shadows, and to improve scene classification and segmentation. For our reproduction we will implement two illumination invariance methods and compare these with the results of the authors. For our reproduction we will use the methods of Maddern et al.[2] and Alvarez et al.[3].

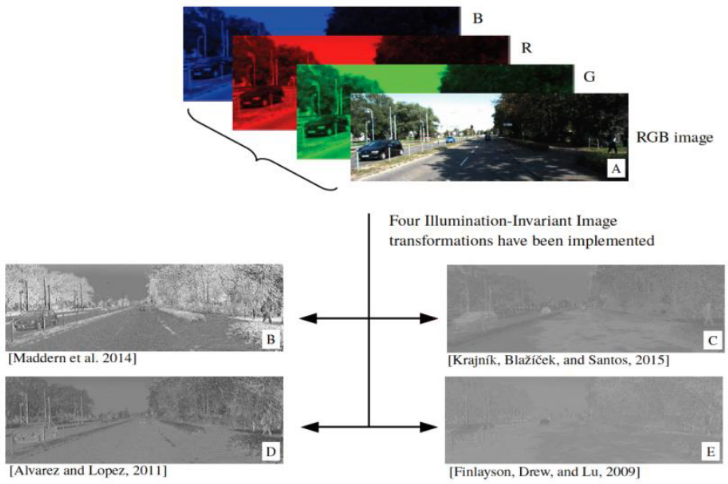


Figure 2: An example of an RGB image followed by four different illumination invariant images, where all the illumination variations such as shadows are significantly reduced within the scenes.

0.3 illumination invariance method - Alvarez et al.

An illumination-invariant image I is a single channel image calculated by combining the three RGB colour channels in the image $IRGB \in \{IR, IG, IB\}$. To compute the illumination invariant images, we use a single channel feature space I combined with three linear sensors $\{R, G, B\}$ as follows for the method of Alvarez et al:

$$I_{\text{Alvarez}} = \cos(\theta) \log_{\text{approx}} \left(\frac{I_R}{I_B}\right) + \sin(\theta) \log_{\text{approx}} \left(\frac{I_G}{I_B}\right)$$

Where I_R, $[G, IB$ are the tree RGB channels, $\theta \in \{0...180\}$, and logapprox 0 is the logarithmic approximation for t $\in \{R, G, B\}$, which is computes as follows:

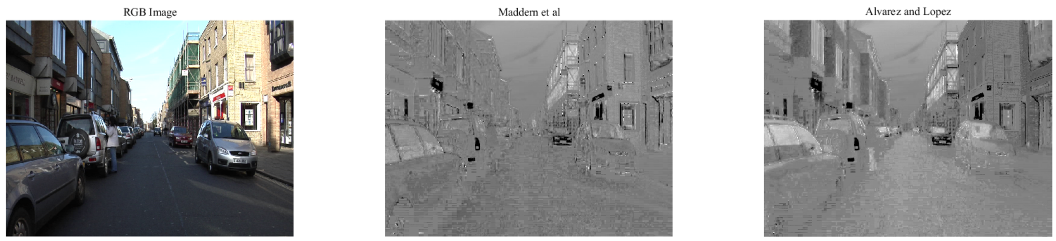
$$\log_{\text{approx}(x)} = \propto \left(\left(x^{\frac{1}{\propto}}\right) - 1\right)$$

Where x is the value from dividing the two channels and $\alpha = 5000$. After evaluating images we decide projection angle $\theta = 135$ degree.## illumination invariance method - Maddern et al. To compute the image for this method, we again convert the 3-channel floating point RGB image into corresponding illumination invariant image as follows:

$$I_{\text{Maddern}} = 0.5 + \log\left(I_G\right) - \propto \log\left(I_B\right) - (1 - \propto) \log\left(I_R\right)$$

Where $\alpha = 0.48$. This illumination-invariant approach was proposed to improve visual localization, mapping and scene classification for autonomous road vehicles.

Our results for image processing:



0.4 How will we evaluate the performance of using these illumination invariant image representations?

We will evaluate the performance of automotive scene understanding and segmentation using the SegNet [4] CNN architecture (Figure 3) with the two aforementioned illumination-invariant transformations.

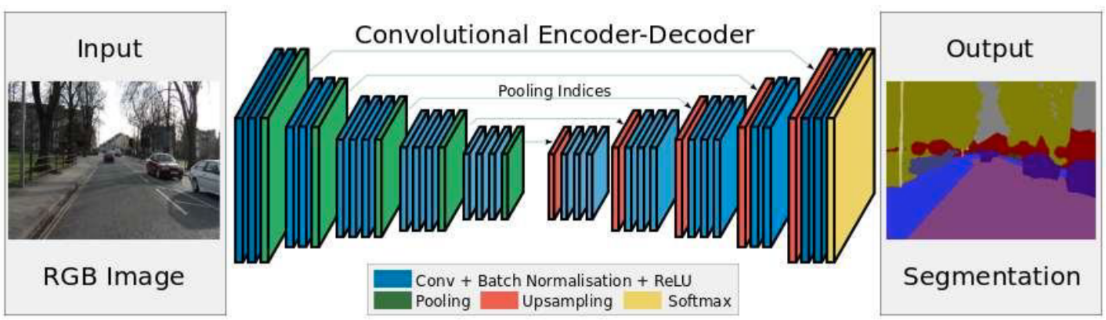


Figure 3: Architecture of the SegNet Convolutional Neural Network

As mentioned before we will use the CamVid dataset with different pixel classes for the SegNet classification task (Figure 1 shows some examples). The authors of the paper used eleven classes: {sky, building, pole, road, pavement, tree, sign, fence, car, pedestrian, bicycle}. The dataset consists of 600 images in total, which we have divided into training, test and validation sets.

0.5 Training the SegNet CNN model

The authors used a VGG16 [5] network pre-trained on the ImageNet [6] dataset, which is the encoder network within SegNet. An encoder network consists of convolution and pooling layers followed by a decoder network containing convolutional and upsampling layers. The authors have used Stochastic Gradient Descent (SGD) optimization, we however have chosen to use Adam optimization in the hopes of getting better results than the authors. Adam can be looked at as a combination of RMSprop and Stochastic Gradient Descent with momentum. It uses the squared gradients to scale the learning rate like RMSprop and it takes advantage of momentum by using moving average of the gradient instead of gradient itself like SGD with momentum[7]. We used the following hyperparameters: initial learning rate 1x10-3, no weight decay 5x10-4 and momentum 0.9. We train the model using the freely provided GPU from Google Colab. For training the neural network on the whole dataset, because of the computation capactiy, we downsized orginal images with scale ratio 0.1. The image resolution then become 72x96, which is 1/5 smaller than the original data. The amount of images for tain, test and validation sets are 447,58,196 respectively, which has approximate proportion of 0.6,0.1,0.3. Self-developed dataloder is used to separate, label processing and transform raw images.

0.6 Code implementation

[1]:
```
# Import Libraries
from __future__ import print_function
import shutil
import os
import torch
import torchvision.transforms as transforms
import torchvision.datasets as datasets
import torchvision.models as models
import torch.nn as nn
import torch.optim as optim
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
%matplotlib inline
import math
from collections import OrderedDict
import torch.nn.functional as F
import pprint
import torch.utils.data as data
import sys
sys.path.append('./')
import utils
import random
```

[2]:
```
# Define dataloader class
class CamVid(data.Dataset):
    """CamVid dataset loader where the dataset is arranged as in
    https://github.com/alexgkendall/SegNet-Tutorial/tree/master/CamVid.
    Keyword arguments:
    - root_dir (``string``): Root directory path.
    - mode (``string``): The type of dataset: 'train' for training set, 'val'
    for validation set, and 'test' for test set.
    - transform (``callable``, optional): A function/transform that  takes in
    an PIL image and returns a transformed version. Default: None.
    - label_transform (``callable``, optional): A function/transform that takes
    in the target and transforms it. Default: None.
    - loader (``callable``, optional): A function to load an image given its
    path. By default ``default_loader`` is used.
```

```
    """
    # Training dataset root folders
    train_folder = 'train'
    train_lbl_folder = 'trainannot'

    # Validation dataset root folders
    val_folder = 'val'
    val_lbl_folder = 'valannot'

    # Test dataset root folders
    test_folder = 'test'
    test_lbl_folder = 'testannot'

    # Images extension
    img_extension = '.png'

    def __init__(self,
                 root_dir,
                 mode='train',
                 transform=None,
                 label_transform=None,
                 loader=utils.pil_loader):

        self.root_dir = root_dir
        self.mode = mode
        self.transform = transform
        self.label_transform = label_transform
        self.loader = loader

        if self.mode.lower() == 'train':
            # Get the training data and labels filepaths
            self.train_data = utils.get_files(
                os.path.join(root_dir, self.train_folder),
                extension_filter=self.img_extension)

            self.train_labels = utils.get_files(
                os.path.join(root_dir, self.train_lbl_folder),
                extension_filter=self.img_extension)
        elif self.mode.lower() == 'val':
            # Get the validation data and labels filepaths
            self.val_data = utils.get_files(
                os.path.join(root_dir, self.val_folder),
                extension_filter=self.img_extension)

            self.val_labels = utils.get_files(
                os.path.join(root_dir, self.val_lbl_folder),
                extension_filter=self.img_extension)
```

```python
        elif self.mode.lower() == 'test':
            # Get the test data and labels filepaths
            self.test_data = utils.get_files(
                os.path.join(root_dir, self.test_folder),
                extension_filter=self.img_extension)

            self.test_labels = utils.get_files(
                os.path.join(root_dir, self.test_lbl_folder),
                extension_filter=self.img_extension)
        else:
            raise RuntimeError("Unexpected dataset mode. "
                               "Supported modes are: train, val and test")

    def __getitem__(self, index):
        """
        Args:
        - index (``int``): index of the item in the dataset
        Returns:
        A tuple of ``PIL.Image`` (image, label) where label is the ground-truth
        of the image.
        """
        if self.mode.lower() == 'train':
            data_path, label_path = self.train_data[index], self.train_labels[
                index]
        elif self.mode.lower() == 'val':
            data_path, label_path = self.val_data[index], self.val_labels[
                index]
        elif self.mode.lower() == 'test':
            data_path, label_path = self.test_data[index], self.test_labels[
                index]
        else:
            raise RuntimeError("Unexpected dataset mode. "
                               "Supported modes are: train, val and test")

        img, label = self.loader(data_path, label_path)
        if self.transform is not None:
            img = self.transform(img)

        if self.label_transform is not None:
            label = self.label_transform(label)
        label = label.permute(1,2,0)*255
        label_np = label.numpy()
        label = rgb_to_label(label_np, colormap=color_encoding)
        label = torch.LongTensor(label)
        return img, label

    def __len__(self):
```

```
        """Returns the length of the dataset."""
        if self.mode.lower() == 'train':
            return len(self.train_data)
        elif self.mode.lower() == 'val':
            return len(self.val_data)
        elif self.mode.lower() == 'test':
            return len(self.test_data)
        else:
            raise RuntimeError("Unexpected dataset mode. "
                               "Supported modes are: train, val and test")
```

[3]:
```
# Default encoding for pixel value, class name, and class color
class_name = {0: 'sky',
      1: 'building',
      2: 'pole',
      3: 'road_marking',
      4: 'road',
      5: 'pavement',
      6: 'tree',
      7: 'sign_symbol',
      8: 'fence',
      9: 'car',
      10: 'pedestrian',
      11: 'bicyclist',
      12: 'unlabeled'}


color_encoding = {0: (128, 128, 128),
      1: (128, 0, 0),
      2: (192, 192, 128),
      3: (255, 69, 0),
      4: (128, 64, 128),
      5: (60, 40, 222),
      6: (128, 128, 0),
      7: (192, 128, 128),
      8: (64, 64, 128),
      9: (64, 0, 128),
      10: (64, 64, 0),
      11: (0, 128, 192),
      12: (0, 0, 0)}

def rgb_to_label(rgb_image, colormap = color_encoding):
    '''Function to one hot encode RGB mask labels
        Inputs:
            rgb_image - image matrix (eg. 256 x 256 x 3 dimension numpy ndarray)
            colormap - dictionary of color to label id
```

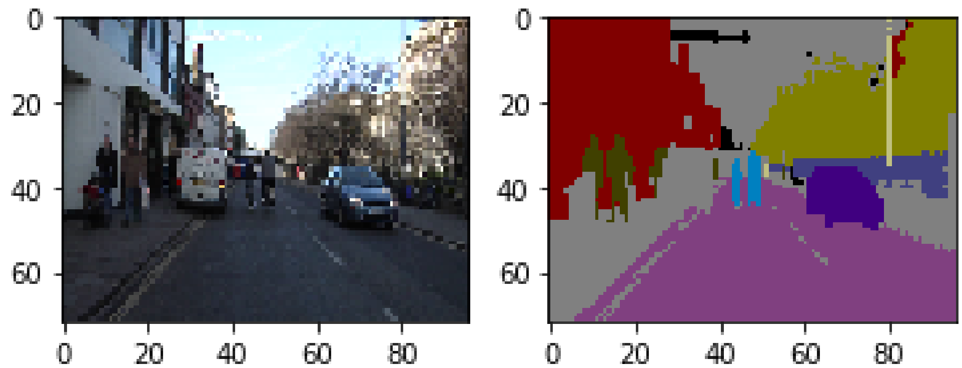
```
        Output: One hot encoded image of dimensions (height x width x␣
↪num_classes) where num_classes = len(colormap)
    '''
    num_classes = len(colormap)
    shape = rgb_image.shape[:2]
    encoded_image = np.zeros(shape,dtype=np.int8)
    for i, cls in enumerate(colormap):
        for x in range(encoded_image.shape[0]):
            for y in range (encoded_image.shape[1]):
                if(np.all(rgb_image[x][y] == colormap[i])):
                    encoded_image[x][y] = i
    return encoded_image


def label_to_rgb(label, colormap = color_encoding):
    '''Function to decode encoded mask labels
        Inputs:
            onehot - one hot encoded image matrix (height x width x num_classes)
            colormap - dictionary of color to label id
        Output: Decoded RGB image (height x width x 3)
    '''
    output = np.zeros(label.shape[:2]+(3,))
    for k in colormap.keys():
        output[label==k] = colormap[k]
    return np.uint8(output)
```

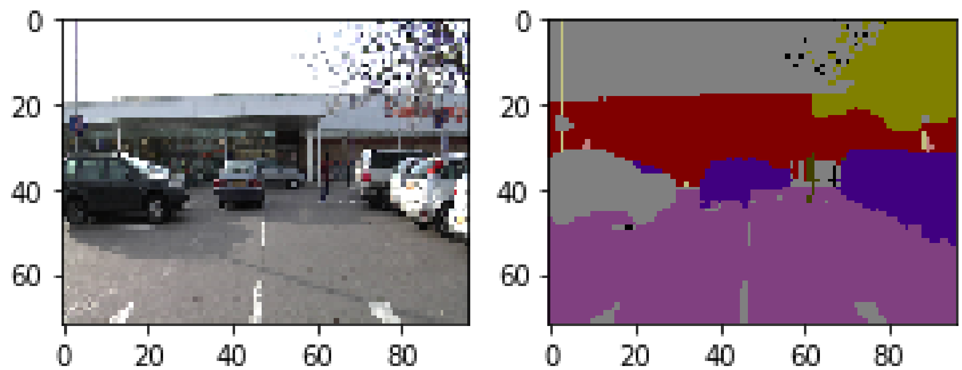
[4]:

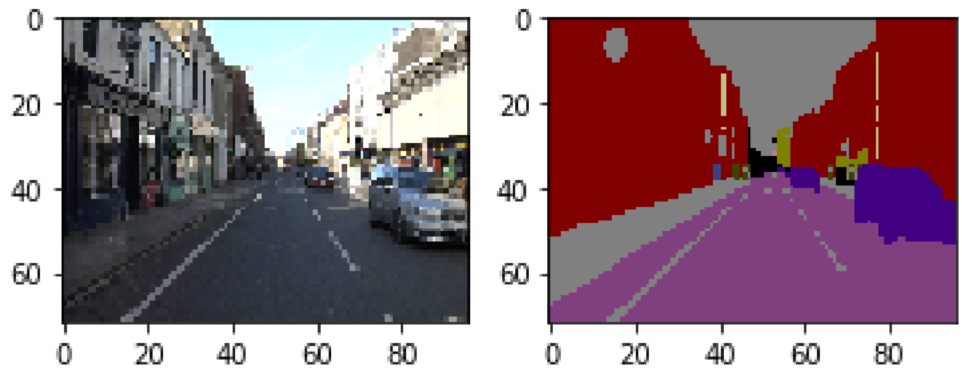
```
# Specify transforms using torchvision.transforms as transforms

res_ratio = 0.1 # set image size

transformations = transforms.Compose([
    transforms.
↪Resize((int(res_ratio*720),int(res_ratio*960)),interpolation=Image.NEAREST),␣
↪#set resolution with nearest neighbor interpolation
    transforms.ToTensor() # Normalize data to be of values [0-1]
])

train = CamVid('./','train',transformations, transformations)
val = CamVid('./','val',transformations, transformations)
test = CamVid('./','test',transformations, transformations)
train_dataloader = data.DataLoader(train, batch_size= 4, shuffle = True,␣
↪num_workers=0)
val_dataloader = data.DataLoader(val, batch_size= 3, shuffle = True,␣
↪num_workers=0)
test_dataloader = data.DataLoader(test, batch_size= 3, shuffle = True,␣
↪num_workers=0)
```

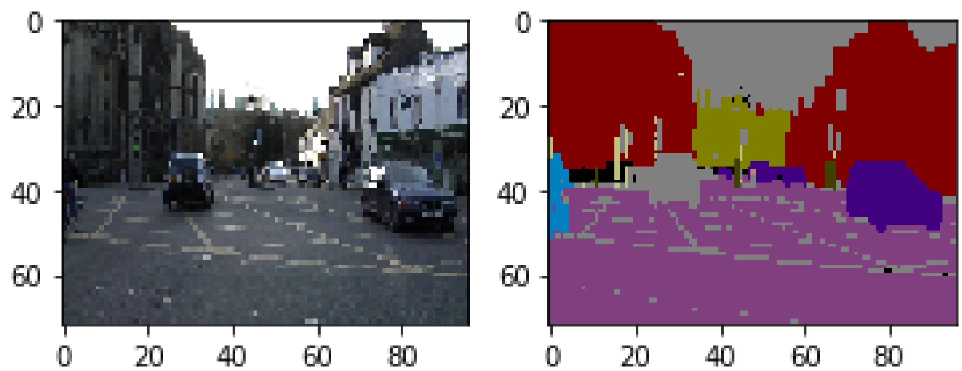
```python
print(len(train_dataloader), len(val_dataloader), len(test_dataloader))

for index,[img,label] in enumerate(train_dataloader):
    print(img.size())
    print(label.size())
    mask_encoded = [label_to_rgb(label[x,:,:].numpy(), color_encoding) for x in range(label.shape[0])]
    for i in range(0,4):
        plt.figure()
        plt.subplot(1,2,1)
        plt.imshow(img[i].permute(1,2,0))
        plt.subplot(1,2,2)
        plt.imshow(mask_encoded[i])
    break
```

```
112 66 20
torch.Size([4, 3, 72, 96])
torch.Size([4, 72, 96])
```









[5]:
```
# SegNet Model - CNN


DEBUG = False


vgg16_dims = [
            (64, 64, 'M'),
            (128, 128, 'M'),
            (256, 256, 256,'M'),
            (512, 512, 512, 'M'),
            (512, 512, 512, 'M')
          ]

decoder_dims = [
             ('U', 512, 512, 512),
             ('U', 512, 512, 512),
             ('U', 256, 256, 256),
             ('U', 128, 128),
```

```
          ('U', 64, 64)
         ]


class SegNet(nn.Module):
    def __init__(self, input_channels, output_channels):
        super(SegNet, self).__init__()

        self.input_channels = input_channels
        self.output_channels = output_channels

        self.vgg16 = models.vgg16(pretrained=True)


        # Encoder layers

        self.encoder_conv_00 = nn.Sequential(*[
                                    nn.Conv2d(in_channels=self.input_channels,
                                    out_channels=64,
                                    kernel_size=3,
                                    padding=1),
                                    nn.BatchNorm2d(64)
                                    ])
        self.encoder_conv_01 = nn.Sequential(*[
                                    nn.Conv2d(in_channels=64,
                                    out_channels=64,
                                    kernel_size=3,
                                    padding=1),
                                    nn.BatchNorm2d(64)
                                   ])
        self.encoder_conv_10 = nn.Sequential(*[
                                    nn.Conv2d(in_channels=64,
                                    out_channels=128,
                                    kernel_size=3,
                                    padding=1),
                                    nn.BatchNorm2d(128)
                                    ])
        self.encoder_conv_11 = nn.Sequential(*[
                                    nn.Conv2d(in_channels=128,
                                    out_channels=128,
                                    kernel_size=3,
                                    padding=1),
                                    nn.BatchNorm2d(128)
                                    ])
        self.encoder_conv_20 = nn.Sequential(*[
                                    nn.Conv2d(in_channels=128,
                                    out_channels=256,
```

```
                                kernel_size=3,
                                padding=1),
                                nn.BatchNorm2d(256)
                                ])
        self.encoder_conv_21 = nn.Sequential(*[
                                nn.Conv2d(in_channels=256,
                                out_channels=256,
                                kernel_size=3,
                                padding=1),
                                nn.BatchNorm2d(256)
                                ])
        self.encoder_conv_22 = nn.Sequential(*[
                                nn.Conv2d(in_channels=256,
                                out_channels=256,
                                kernel_size=3,
                                padding=1),
                                nn.BatchNorm2d(256)
                                ])
        self.encoder_conv_30 = nn.Sequential(*[
                                nn.Conv2d(in_channels=256,
                                out_channels=512,
                                kernel_size=3,
                                padding=1),
                                nn.BatchNorm2d(512)
                                ])
        self.encoder_conv_31 = nn.Sequential(*[
                                nn.Conv2d(in_channels=512,
                                out_channels=512,
                                kernel_size=3,
                                padding=1),
                                nn.BatchNorm2d(512)
                                ])
        self.encoder_conv_32 = nn.Sequential(*[
                                nn.Conv2d(in_channels=512,
                                out_channels=512,
                                kernel_size=3,
                                padding=1),
                                nn.BatchNorm2d(512)
                                ])
        self.encoder_conv_40 = nn.Sequential(*[
                                nn.Conv2d(in_channels=512,
                                out_channels=512,
                                kernel_size=3,
                                padding=1),
                                nn.BatchNorm2d(512)
                                ])
        self.encoder_conv_41 = nn.Sequential(*[
```

```
                          nn.Conv2d(in_channels=512,
                          out_channels=512,
                          kernel_size=3,
                          padding=1),
                          nn.BatchNorm2d(512)
                          ])
        self.encoder_conv_42 = nn.Sequential(*[
                          nn.Conv2d(in_channels=512,
                          out_channels=512,
                          kernel_size=3,
                          padding=1),
                          nn.BatchNorm2d(512)
                          ])

        self.init_vgg_weigts()

        # Decoder layers

        self.decoder_convtr_42 = nn.Sequential(*[
                           nn.ConvTranspose2d(in_channels=512,
                           out_channels=512,
                           kernel_size=3,
                           padding=1),
                           nn.BatchNorm2d(512)
                           ])
        self.decoder_convtr_41 = nn.Sequential(*[
                           nn.ConvTranspose2d(in_channels=512,
                           out_channels=512,
                           kernel_size=3,
                           padding=1),
                           nn.BatchNorm2d(512)
                           ])
        self.decoder_convtr_40 = nn.Sequential(*[
                           nn.ConvTranspose2d(in_channels=512,
                           out_channels=512,
                           kernel_size=3,
                           padding=1),
                           nn.BatchNorm2d(512)
                           ])
        self.decoder_convtr_32 = nn.Sequential(*[
                           nn.ConvTranspose2d(in_channels=512,
                           out_channels=512,
                           kernel_size=3,
                           padding=1),
                           nn.BatchNorm2d(512)
                           ])
        self.decoder_convtr_31 = nn.Sequential(*[
```

```
                            nn.ConvTranspose2d(in_channels=512,
                            out_channels=512,
                            kernel_size=3,
                            padding=1),
                            nn.BatchNorm2d(512)
                            ])
        self.decoder_convtr_30 = nn.Sequential(*[
                            nn.ConvTranspose2d(in_channels=512,
                            out_channels=256,
                            kernel_size=3,
                            padding=1),
                            nn.BatchNorm2d(256)
                            ])
        self.decoder_convtr_22 = nn.Sequential(*[
                            nn.ConvTranspose2d(in_channels=256,
                            out_channels=256,
                            kernel_size=3,
                            padding=1),
                            nn.BatchNorm2d(256)
                            ])
        self.decoder_convtr_21 = nn.Sequential(*[
                            nn.ConvTranspose2d(in_channels=256,
                            out_channels=256,
                            kernel_size=3,
                            padding=1),
                            nn.BatchNorm2d(256)
                            ])
        self.decoder_convtr_20 = nn.Sequential(*[
                            nn.ConvTranspose2d(in_channels=256,
                            out_channels=128,
                            kernel_size=3,
                            padding=1),
                            nn.BatchNorm2d(128)
                            ])
        self.decoder_convtr_11 = nn.Sequential(*[
                            nn.ConvTranspose2d(in_channels=128,
                            out_channels=128,
                            kernel_size=3,
                            padding=1),
                            nn.BatchNorm2d(128)
                            ])
        self.decoder_convtr_10 = nn.Sequential(*[
                            nn.ConvTranspose2d(in_channels=128,
                            out_channels=64,
                            kernel_size=3,
                            padding=1),
                            nn.BatchNorm2d(64)
```

```
                              ])
        self.decoder_convtr_01 = nn.Sequential(*[
                              nn.ConvTranspose2d(in_channels=64,
                              out_channels=64,
                              kernel_size=3,
                              padding=1),
                              nn.BatchNorm2d(64)
                              ])
        self.decoder_convtr_00 = nn.Sequential(*[
                              nn.ConvTranspose2d(in_channels=64,
                              out_channels=self.output_channels,
                              kernel_size=3,
                              padding=1)
                              ])


    def forward(self, input_img):
        """
        Forward pass `input_img` through the network
        """

        # Encoder

        # Encoder Stage - 1
        dim_0 = input_img.size()
        x_00 = F.relu(self.encoder_conv_00(input_img))
        x_01 = F.relu(self.encoder_conv_01(x_00))
        x_0, indices_0 = F.max_pool2d(x_01, kernel_size=2, stride=2,␣
↪return_indices=True)

        # Encoder Stage - 2
        dim_1 = x_0.size()
        x_10 = F.relu(self.encoder_conv_10(x_0))
        x_11 = F.relu(self.encoder_conv_11(x_10))
        x_1, indices_1 = F.max_pool2d(x_11, kernel_size=2, stride=2,␣
↪return_indices=True)

        # Encoder Stage - 3
        dim_2 = x_1.size()
        x_20 = F.relu(self.encoder_conv_20(x_1))
        x_21 = F.relu(self.encoder_conv_21(x_20))
        x_22 = F.relu(self.encoder_conv_22(x_21))
        x_2, indices_2 = F.max_pool2d(x_22, kernel_size=2, stride=2,␣
↪return_indices=True)

        # Encoder Stage - 4
        dim_3 = x_2.size()
```

```
        x_30 = F.relu(self.encoder_conv_30(x_2))
        x_31 = F.relu(self.encoder_conv_31(x_30))
        x_32 = F.relu(self.encoder_conv_32(x_31))
        x_3, indices_3 = F.max_pool2d(x_32, kernel_size=2, stride=2,␣
↪return_indices=True)

        # Encoder Stage - 5
        dim_4 = x_3.size()
        x_40 = F.relu(self.encoder_conv_40(x_3))
        x_41 = F.relu(self.encoder_conv_41(x_40))
        x_42 = F.relu(self.encoder_conv_42(x_41))
        x_4, indices_4 = F.max_pool2d(x_42, kernel_size=2, stride=2,␣
↪return_indices=True)

        # Decoder

        dim_d = x_4.size()

        # Decoder Stage - 5
        x_4d = F.max_unpool2d(x_4, indices_4, kernel_size=2, stride=2,␣
↪output_size=dim_4)
        x_42d = F.relu(self.decoder_convtr_42(x_4d))
        x_41d = F.relu(self.decoder_convtr_41(x_42d))
        x_40d = F.relu(self.decoder_convtr_40(x_41d))
        dim_4d = x_40d.size()

        # Decoder Stage - 4
        x_3d = F.max_unpool2d(x_40d, indices_3, kernel_size=2, stride=2,␣
↪output_size=dim_3)
        x_32d = F.relu(self.decoder_convtr_32(x_3d))
        x_31d = F.relu(self.decoder_convtr_31(x_32d))
        x_30d = F.relu(self.decoder_convtr_30(x_31d))
        dim_3d = x_30d.size()

        # Decoder Stage - 3
        x_2d = F.max_unpool2d(x_30d, indices_2, kernel_size=2, stride=2,␣
↪output_size=dim_2)
        x_22d = F.relu(self.decoder_convtr_22(x_2d))
        x_21d = F.relu(self.decoder_convtr_21(x_22d))
        x_20d = F.relu(self.decoder_convtr_20(x_21d))
        dim_2d = x_20d.size()

        # Decoder Stage - 2
        x_1d = F.max_unpool2d(x_20d, indices_1, kernel_size=2, stride=2,␣
↪output_size=dim_1)
        x_11d = F.relu(self.decoder_convtr_11(x_1d))
```

```
        x_10d = F.relu(self.decoder_convtr_10(x_11d))
        dim_1d = x_10d.size()

        # Decoder Stage - 1
        x_0d = F.max_unpool2d(x_10d, indices_0, kernel_size=2, stride=2,␣
↪output_size=dim_0)
        x_01d = F.relu(self.decoder_convtr_01(x_0d))
        x_00d = self.decoder_convtr_00(x_01d)
        dim_0d = x_00d.size()

        x_softmax = F.softmax(x_00d, dim=1)


        if DEBUG:
            print("dim_0: {}".format(dim_0))
            print("dim_1: {}".format(dim_1))
            print("dim_2: {}".format(dim_2))
            print("dim_3: {}".format(dim_3))
            print("dim_4: {}".format(dim_4))

            print("dim_d: {}".format(dim_d))
            print("dim_4d: {}".format(dim_4d))
            print("dim_3d: {}".format(dim_3d))
            print("dim_2d: {}".format(dim_2d))
            print("dim_1d: {}".format(dim_1d))
            print("dim_0d: {}".format(dim_0d))


        return x_00d, x_softmax


    def init_vgg_weigts(self):
        assert self.encoder_conv_00[0].weight.size() == self.vgg16.features[0].
↪weight.size()
        self.encoder_conv_00[0].weight.data = self.vgg16.features[0].weight.data
        assert self.encoder_conv_00[0].bias.size() == self.vgg16.features[0].
↪bias.size()
        self.encoder_conv_00[0].bias.data = self.vgg16.features[0].bias.data

        assert self.encoder_conv_01[0].weight.size() == self.vgg16.features[2].
↪weight.size()
        self.encoder_conv_01[0].weight.data = self.vgg16.features[2].weight.data
        assert self.encoder_conv_01[0].bias.size() == self.vgg16.features[2].
↪bias.size()
        self.encoder_conv_01[0].bias.data = self.vgg16.features[2].bias.data
```

```
        assert self.encoder_conv_10[0].weight.size() == self.vgg16.features[5].
↪weight.size()
        self.encoder_conv_10[0].weight.data = self.vgg16.features[5].weight.data
        assert self.encoder_conv_10[0].bias.size() == self.vgg16.features[5].
↪bias.size()
        self.encoder_conv_10[0].bias.data = self.vgg16.features[5].bias.data

        assert self.encoder_conv_11[0].weight.size() == self.vgg16.features[7].
↪weight.size()
        self.encoder_conv_11[0].weight.data = self.vgg16.features[7].weight.data
        assert self.encoder_conv_11[0].bias.size() == self.vgg16.features[7].
↪bias.size()
        self.encoder_conv_11[0].bias.data = self.vgg16.features[7].bias.data

        assert self.encoder_conv_20[0].weight.size() == self.vgg16.features[10].
↪weight.size()
        self.encoder_conv_20[0].weight.data = self.vgg16.features[10].weight.data
        assert self.encoder_conv_20[0].bias.size() == self.vgg16.features[10].
↪bias.size()
        self.encoder_conv_20[0].bias.data = self.vgg16.features[10].bias.data

        assert self.encoder_conv_21[0].weight.size() == self.vgg16.features[12].
↪weight.size()
        self.encoder_conv_21[0].weight.data = self.vgg16.features[12].weight.data
        assert self.encoder_conv_21[0].bias.size() == self.vgg16.features[12].
↪bias.size()
        self.encoder_conv_21[0].bias.data = self.vgg16.features[12].bias.data

        assert self.encoder_conv_22[0].weight.size() == self.vgg16.features[14].
↪weight.size()
        self.encoder_conv_22[0].weight.data = self.vgg16.features[14].weight.data
        assert self.encoder_conv_22[0].bias.size() == self.vgg16.features[14].
↪bias.size()
        self.encoder_conv_22[0].bias.data = self.vgg16.features[14].bias.data

        assert self.encoder_conv_30[0].weight.size() == self.vgg16.features[17].
↪weight.size()
        self.encoder_conv_30[0].weight.data = self.vgg16.features[17].weight.data
        assert self.encoder_conv_30[0].bias.size() == self.vgg16.features[17].
↪bias.size()
        self.encoder_conv_30[0].bias.data = self.vgg16.features[17].bias.data

        assert self.encoder_conv_31[0].weight.size() == self.vgg16.features[19].
↪weight.size()
        self.encoder_conv_31[0].weight.data = self.vgg16.features[19].weight.data
```

```
        assert self.encoder_conv_31[0].bias.size() == self.vgg16.features[19].
↪bias.size()
        self.encoder_conv_31[0].bias.data = self.vgg16.features[19].bias.data

        assert self.encoder_conv_32[0].weight.size() == self.vgg16.features[21].
↪weight.size()
        self.encoder_conv_32[0].weight.data = self.vgg16.features[21].weight.data
        assert self.encoder_conv_32[0].bias.size() == self.vgg16.features[21].
↪bias.size()
        self.encoder_conv_32[0].bias.data = self.vgg16.features[21].bias.data

        assert self.encoder_conv_40[0].weight.size() == self.vgg16.features[24].
↪weight.size()
        self.encoder_conv_40[0].weight.data = self.vgg16.features[24].weight.data
        assert self.encoder_conv_40[0].bias.size() == self.vgg16.features[24].
↪bias.size()
        self.encoder_conv_40[0].bias.data = self.vgg16.features[24].bias.data

        assert self.encoder_conv_41[0].weight.size() == self.vgg16.features[26].
↪weight.size()
        self.encoder_conv_41[0].weight.data = self.vgg16.features[26].weight.data
        assert self.encoder_conv_41[0].bias.size() == self.vgg16.features[26].
↪bias.size()
        self.encoder_conv_41[0].bias.data = self.vgg16.features[26].bias.data

        assert self.encoder_conv_42[0].weight.size() == self.vgg16.features[28].
↪weight.size()
        self.encoder_conv_42[0].weight.data = self.vgg16.features[28].weight.data
        assert self.encoder_conv_42[0].bias.size() == self.vgg16.features[28].
↪bias.size()
        self.encoder_conv_42[0].bias.data = self.vgg16.features[28].bias.data
```

Test for a single image

[6]:
```
# Extract a single image and mask
for index,[img,label] in enumerate(train_dataloader):
    x = img
    y = label
    break
```

[87]:
```
"""
Train a SegNet model
Usage:
python train.py --data_root /home/SharedData/intern_sayan/PascalVOC2012/data/
↪VOCdevkit/VOC2012/ \
                --train_path ImageSets/Segmentation/train.txt \
```

```
                --img_dir JPEGImages \
                --mask_dir SegmentationClass \
                --save_dir /home/SharedData/intern_sayan/PascalVOC2012/ \
                --checkpoint /home/SharedData/intern_sayan/PascalVOC2012/
↪model_best.pth \
                --gpu 1
"""

from __future__ import print_function
import os
import time
import torch
from torch.utils.data import DataLoader
from torch.autograd import Variable



# Parameters
NUM_INPUT_CHANNELS = 3
NUM_OUTPUT_CHANNELS = 13

NUM_EPOCHS = 200

LEARNING_RATE = 1e-3
MOMENTUM = 0.9
BATCH_SIZE = 4



def train():
    is_better = True
    prev_loss = float('inf')
#     for continue learning
#     model.load_state_dict(torch.load("./model_best.pth"))
    model.train()

    loss_list = []
    acc_list = []
    for epoch in range(NUM_EPOCHS):
        loss_f = 0
        t_start = time.time()


        input_tensor = Variable(x)
        target_tensor = Variable(y)
        if CUDA:
            input_tensor = input_tensor.cuda()
```

```
            target_tensor = target_tensor.cuda()

        predicted_tensor, softmaxed_tensor = model(input_tensor)


        optimizer.zero_grad()
        loss = criterion(predicted_tensor, target_tensor)
        loss.backward()
        optimizer.step()


        loss_f += loss.float()

        _, prediction = torch.max(predicted_tensor,1)
        y_true = torch.flatten(target_tensor)
        y_prep = torch.flatten(prediction)
        intersection = torch.sum(y_true * y_prep)
        acc = (2. * intersection) / (torch.sum(y_true*y_true) + torch.
 ↪sum(y_prep*y_prep))
        acc = acc.detach().cpu().numpy()
        acc_list.append(acc)

        delta = time.time() - t_start
        is_better = loss_f < prev_loss
        loss_list.append(loss_f/len(x))
        if is_better:
            prev_loss = loss_f
        torch.save(model.state_dict(), os.path.join('./', "model_best.pth"))

        print("Epoch #{}\tLoss: {:.8f}\t Time: {:2f}s".format(epoch+1, loss_f,␣
 ↪delta))
    return loss_list, acc_list

if __name__ == "__main__":


    CUDA = torch.cuda.is_available()

    print(CUDA)
    # train_dataloader = data.DataLoader(train, batch_size=4, shuffle = True,␣
 ↪num_workers=4)

    # weights when using median frequency balancing used in SegNet paper
    # https://arxiv.org/pdf/1511.00561.pdf
    # The numbers were generated by:
    # https://github.com/yandex/segnet-torch/blob/master/datasets/camvid-gen.lua
    CAMVID_CLASS_WEIGHTS = [0.58872014284134,
```

```
                  0.51052379608154,
                  2.6966278553009,
                  0.45021694898605,
                  1.1785038709641,
                  0.77028578519821,
                  2.4782588481903,
                  2.5273461341858,
                  1.0122526884079,
                  3.2375309467316,
                  4.1312313079834,
                  0.3,
                  0]
    if CUDA:
        model = SegNet(input_channels=NUM_INPUT_CHANNELS,␣
↪output_channels=NUM_OUTPUT_CHANNELS).cuda()
        class_weights = torch.FloatTensor(CAMVID_CLASS_WEIGHTS).cuda()
        criterion = torch.nn.CrossEntropyLoss(weight=class_weights).cuda()
    else:
        model = SegNet(input_channels=NUM_INPUT_CHANNELS,␣
↪output_channels=NUM_OUTPUT_CHANNELS)
        class_weights = torch.FloatTensor(CAMVID_CLASS_WEIGHTS)
        criterion = torch.nn.CrossEntropyLoss(weight=class_weights)


    optimizer = torch.optim.Adam(model.parameters(), lr=LEARNING_RATE)

    loss_list = []
    acc_list = []
    loss_list, acc_list = train()
```

```
True
Epoch #1        Loss: 2.55754876        Time: 0.096742s
Epoch #2        Loss: 2.32248020        Time: 0.084774s
Epoch #3        Loss: 1.91855991        Time: 0.079787s
Epoch #4        Loss: 1.53522611        Time: 0.081776s
Epoch #5        Loss: 1.23839629        Time: 0.083776s
Epoch #6        Loss: 1.08583093        Time: 0.081781s
Epoch #7        Loss: 0.95990711        Time: 0.078834s
Epoch #8        Loss: 0.84045833        Time: 0.078817s
Epoch #9        Loss: 0.74290419        Time: 0.082778s
Epoch #10       Loss: 0.65871602        Time: 0.080783s
Epoch #11       Loss: 0.60670245        Time: 0.080816s
Epoch #12       Loss: 0.55186510        Time: 0.083745s
Epoch #13       Loss: 0.47650722        Time: 0.079786s
Epoch #14       Loss: 0.45593768        Time: 0.078789s
Epoch #15       Loss: 0.39415997        Time: 0.077837s
Epoch #16       Loss: 0.37400636        Time: 0.081813s
```

```
Epoch #17        Loss: 0.33640644        Time: 0.079787s
Epoch #18        Loss: 0.31342551        Time: 0.081816s
Epoch #19        Loss: 0.29602915        Time: 0.086768s
Epoch #20        Loss: 0.27636454        Time: 0.083809s
Epoch #21        Loss: 0.25970283        Time: 0.081781s
Epoch #22        Loss: 0.24345094        Time: 0.082803s
Epoch #23        Loss: 0.22949116        Time: 0.081814s
Epoch #24        Loss: 0.21790645        Time: 0.078788s
Epoch #25        Loss: 0.20751855        Time: 0.080784s
Epoch #26        Loss: 0.19523451        Time: 0.081780s
Epoch #27        Loss: 0.18633641        Time: 0.082814s
Epoch #28        Loss: 0.17760716        Time: 0.082778s
Epoch #29        Loss: 0.16844785        Time: 0.082778s
Epoch #30        Loss: 0.16031061        Time: 0.087765s
Epoch #31        Loss: 0.15232986        Time: 0.086764s
Epoch #32        Loss: 0.14553648        Time: 0.080784s
Epoch #33        Loss: 0.13827479        Time: 0.081780s
Epoch #34        Loss: 0.13154685        Time: 0.083775s
Epoch #35        Loss: 0.12555325        Time: 0.078820s
Epoch #36        Loss: 0.11904900        Time: 0.081816s
Epoch #37        Loss: 0.11341165        Time: 0.081808s
Epoch #38        Loss: 0.10758978        Time: 0.080781s
Epoch #39        Loss: 0.10226027        Time: 0.085739s
Epoch #40        Loss: 0.09695361        Time: 0.088763s
Epoch #41        Loss: 0.09219056        Time: 0.081781s
Epoch #42        Loss: 0.08743983        Time: 0.080785s
Epoch #43        Loss: 0.08294282        Time: 0.080822s
Epoch #44        Loss: 0.07861019        Time: 0.080781s
Epoch #45        Loss: 0.07444242        Time: 0.083793s
Epoch #46        Loss: 0.07048448        Time: 0.080784s
Epoch #47        Loss: 0.06666543        Time: 0.080784s
Epoch #48        Loss: 0.06289131        Time: 0.081781s
Epoch #49        Loss: 0.05930397        Time: 0.082810s
Epoch #50        Loss: 0.05600355        Time: 0.081795s
Epoch #51        Loss: 0.05259738        Time: 0.079798s
Epoch #52        Loss: 0.04952695        Time: 0.083776s
Epoch #53        Loss: 0.04655929        Time: 0.083774s
Epoch #54        Loss: 0.04383587        Time: 0.081262s
Epoch #55        Loss: 0.04127233        Time: 0.081781s
Epoch #56        Loss: 0.03889567        Time: 0.085771s
Epoch #57        Loss: 0.03671874        Time: 0.090757s
Epoch #58        Loss: 0.03468009        Time: 0.084805s
Epoch #59        Loss: 0.03283238        Time: 0.080783s
Epoch #60        Loss: 0.03112663        Time: 0.082817s
Epoch #61        Loss: 0.02948867        Time: 0.084773s
Epoch #62        Loss: 0.02804064        Time: 0.092453s
Epoch #63        Loss: 0.02670176        Time: 0.083776s
Epoch #64        Loss: 0.02541379        Time: 0.084774s
```

```
Epoch #65       Loss: 0.02425924        Time: 0.084779s
Epoch #66       Loss: 0.02319516        Time: 0.083797s
Epoch #67       Loss: 0.02210841        Time: 0.079792s
Epoch #68       Loss: 0.02111185        Time: 0.080783s
Epoch #69       Loss: 0.02018378        Time: 0.082779s
Epoch #70       Loss: 0.01928605        Time: 0.088762s
Epoch #71       Loss: 0.01847053        Time: 0.086767s
Epoch #72       Loss: 0.01770151        Time: 0.084773s
Epoch #73       Loss: 0.01694929        Time: 0.079792s
Epoch #74       Loss: 0.01624670        Time: 0.083809s
Epoch #75       Loss: 0.01556245        Time: 0.078789s
Epoch #76       Loss: 0.01491553        Time: 0.086876s
Epoch #77       Loss: 0.01432420        Time: 0.086768s
Epoch #78       Loss: 0.01374433        Time: 0.079786s
Epoch #79       Loss: 0.01321895        Time: 0.087764s
Epoch #80       Loss: 0.01269828        Time: 0.083774s
Epoch #81       Loss: 0.01223124        Time: 0.095731s
Epoch #82       Loss: 0.01178118        Time: 0.081781s
Epoch #83       Loss: 0.01137202        Time: 0.081781s
Epoch #84       Loss: 0.01094687        Time: 0.090758s
Epoch #85       Loss: 0.01059698        Time: 0.080828s
Epoch #86       Loss: 0.01023815        Time: 0.082778s
Epoch #87       Loss: 0.00987351        Time: 0.087767s
Epoch #88       Loss: 0.00953647        Time: 0.078790s
Epoch #89       Loss: 0.00922411        Time: 0.086737s
Epoch #90       Loss: 0.00892886        Time: 0.080750s
Epoch #91       Loss: 0.00864273        Time: 0.080784s
Epoch #92       Loss: 0.00837226        Time: 0.081746s
Epoch #93       Loss: 0.00811769        Time: 0.079819s
Epoch #94       Loss: 0.00787541        Time: 0.083777s
Epoch #95       Loss: 0.00763886        Time: 0.080818s
Epoch #96       Loss: 0.00742651        Time: 0.084771s
Epoch #97       Loss: 0.00720533        Time: 0.085771s
Epoch #98       Loss: 0.00700867        Time: 0.083775s
Epoch #99       Loss: 0.00681655        Time: 0.092753s
Epoch #100      Loss: 0.00663306        Time: 0.083776s
Epoch #101      Loss: 0.00645724        Time: 0.084773s
Epoch #102      Loss: 0.00629514        Time: 0.084062s
Epoch #103      Loss: 0.00613045        Time: 0.082851s
Epoch #104      Loss: 0.00597936        Time: 0.081829s
Epoch #105      Loss: 0.00583058        Time: 0.081806s
Epoch #106      Loss: 0.00570537        Time: 0.083773s
Epoch #107      Loss: 0.00556377        Time: 0.080783s
Epoch #108      Loss: 0.00543111        Time: 0.079842s
Epoch #109      Loss: 0.00530550        Time: 0.085768s
Epoch #110      Loss: 0.00518038        Time: 0.078821s
Epoch #111      Loss: 0.00506751        Time: 0.084774s
Epoch #112      Loss: 0.00496021        Time: 0.082778s
```

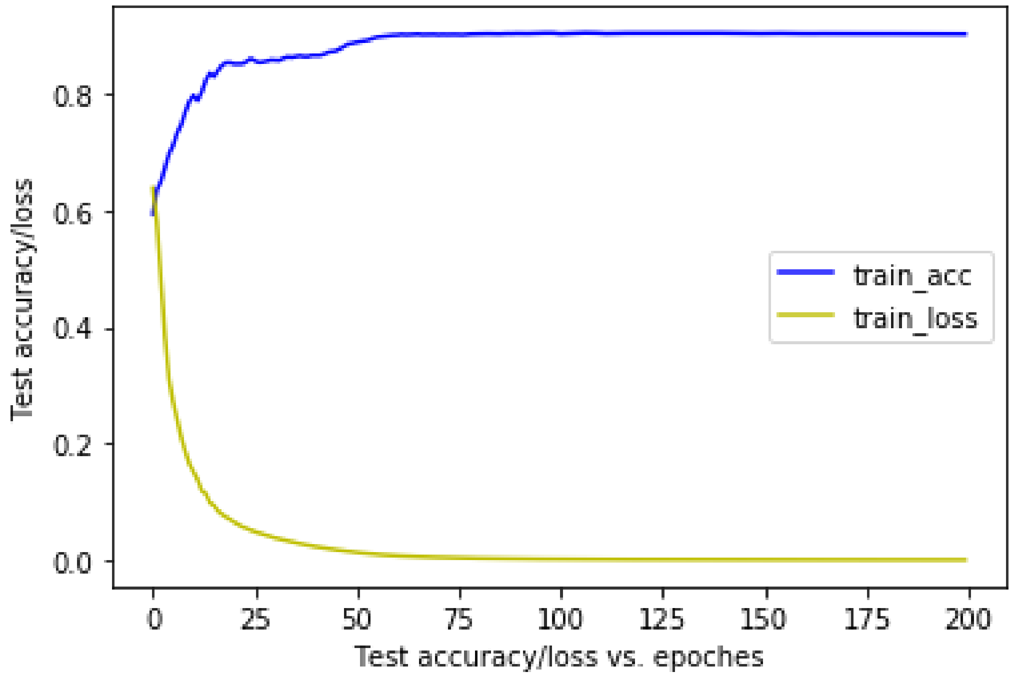
```
Epoch #113      Loss: 0.00485127      Time: 0.082747s
Epoch #114      Loss: 0.00475028      Time: 0.084772s
Epoch #115      Loss: 0.00465720      Time: 0.081786s
Epoch #116      Loss: 0.00456407      Time: 0.080783s
Epoch #117      Loss: 0.00447794      Time: 0.083775s
Epoch #118      Loss: 0.00438806      Time: 0.082793s
Epoch #119      Loss: 0.00429752      Time: 0.078799s
Epoch #120      Loss: 0.00420985      Time: 0.082778s
Epoch #121      Loss: 0.00412936      Time: 0.086768s
Epoch #122      Loss: 0.00405786      Time: 0.079787s
Epoch #123      Loss: 0.00398485      Time: 0.080796s
Epoch #124      Loss: 0.00391647      Time: 0.082778s
Epoch #125      Loss: 0.00385709      Time: 0.081770s
Epoch #126      Loss: 0.00379302      Time: 0.082790s
Epoch #127      Loss: 0.00372220      Time: 0.080815s
Epoch #128      Loss: 0.00365963      Time: 0.081781s
Epoch #129      Loss: 0.00359425      Time: 0.085771s
Epoch #130      Loss: 0.00353090      Time: 0.083776s
Epoch #131      Loss: 0.00347060      Time: 0.080783s
Epoch #132      Loss: 0.00341160      Time: 0.079787s
Epoch #133      Loss: 0.00335626      Time: 0.079786s
Epoch #134      Loss: 0.00330193      Time: 0.084772s
Epoch #135      Loss: 0.00325376      Time: 0.082827s
Epoch #136      Loss: 0.00319987      Time: 0.079785s
Epoch #137      Loss: 0.00314772      Time: 0.083744s
Epoch #138      Loss: 0.00309894      Time: 0.080817s
Epoch #139      Loss: 0.00306179      Time: 0.083807s
Epoch #140      Loss: 0.00301197      Time: 0.078821s
Epoch #141      Loss: 0.00296639      Time: 0.079785s
Epoch #142      Loss: 0.00292266      Time: 0.083803s
Epoch #143      Loss: 0.00288073      Time: 0.080814s
Epoch #144      Loss: 0.00284072      Time: 0.079820s
Epoch #145      Loss: 0.00280011      Time: 0.082780s
Epoch #146      Loss: 0.00276002      Time: 0.082755s
Epoch #147      Loss: 0.00272262      Time: 0.078789s
Epoch #148      Loss: 0.00268664      Time: 0.078833s
Epoch #149      Loss: 0.00264995      Time: 0.077792s
Epoch #150      Loss: 0.00261384      Time: 0.080812s
Epoch #151      Loss: 0.00257895      Time: 0.078821s
Epoch #152      Loss: 0.00254511      Time: 0.083787s
Epoch #153      Loss: 0.00251233      Time: 0.078822s
Epoch #154      Loss: 0.00248032      Time: 0.088762s
Epoch #155      Loss: 0.00244892      Time: 0.079787s
Epoch #156      Loss: 0.00241803      Time: 0.081781s
Epoch #157      Loss: 0.00238798      Time: 0.078832s
Epoch #158      Loss: 0.00235874      Time: 0.080783s
Epoch #159      Loss: 0.00232989      Time: 0.083807s
Epoch #160      Loss: 0.00230183      Time: 0.082744s
```

```
Epoch #161      Loss: 0.00227423        Time: 0.087734s
Epoch #162      Loss: 0.00224787        Time: 0.081815s
Epoch #163      Loss: 0.00222185        Time: 0.078758s
Epoch #164      Loss: 0.00219617        Time: 0.081782s
Epoch #165      Loss: 0.00217079        Time: 0.078789s
Epoch #166      Loss: 0.00214699        Time: 0.078789s
Epoch #167      Loss: 0.00212278        Time: 0.083774s
Epoch #168      Loss: 0.00209864        Time: 0.080780s
Epoch #169      Loss: 0.00207498        Time: 0.081781s
Epoch #170      Loss: 0.00205225        Time: 0.078822s
Epoch #171      Loss: 0.00202994        Time: 0.080783s
Epoch #172      Loss: 0.00200800        Time: 0.080783s
Epoch #173      Loss: 0.00198646        Time: 0.082779s
Epoch #174      Loss: 0.00196560        Time: 0.082778s
Epoch #175      Loss: 0.00194497        Time: 0.087766s
Epoch #176      Loss: 0.00192446        Time: 0.081813s
Epoch #177      Loss: 0.00190439        Time: 0.079787s
Epoch #178      Loss: 0.00188466        Time: 0.080804s
Epoch #179      Loss: 0.00186683        Time: 0.081781s
Epoch #180      Loss: 0.00184656        Time: 0.086768s
Epoch #181      Loss: 0.00182857        Time: 0.079755s
Epoch #182      Loss: 0.00181027        Time: 0.079786s
Epoch #183      Loss: 0.00179571        Time: 0.085804s
Epoch #184      Loss: 0.00177862        Time: 0.081804s
Epoch #185      Loss: 0.00176072        Time: 0.080816s
Epoch #186      Loss: 0.00174336        Time: 0.080817s
Epoch #187      Loss: 0.00172666        Time: 0.082813s
Epoch #188      Loss: 0.00170992        Time: 0.077755s
Epoch #189      Loss: 0.00169344        Time: 0.080758s
Epoch #190      Loss: 0.00167738        Time: 0.077791s
Epoch #191      Loss: 0.00166144        Time: 0.079785s
Epoch #192      Loss: 0.00164748        Time: 0.083792s
Epoch #193      Loss: 0.00163225        Time: 0.080816s
Epoch #194      Loss: 0.00161725        Time: 0.081782s
Epoch #195      Loss: 0.00160575        Time: 0.086760s
Epoch #196      Loss: 0.00159144        Time: 0.083777s
Epoch #197      Loss: 0.00157649        Time: 0.080752s
Epoch #198      Loss: 0.00156168        Time: 0.087765s
Epoch #199      Loss: 0.00154735        Time: 0.082826s
Epoch #200      Loss: 0.00153315        Time: 0.077825s
```

[97]:

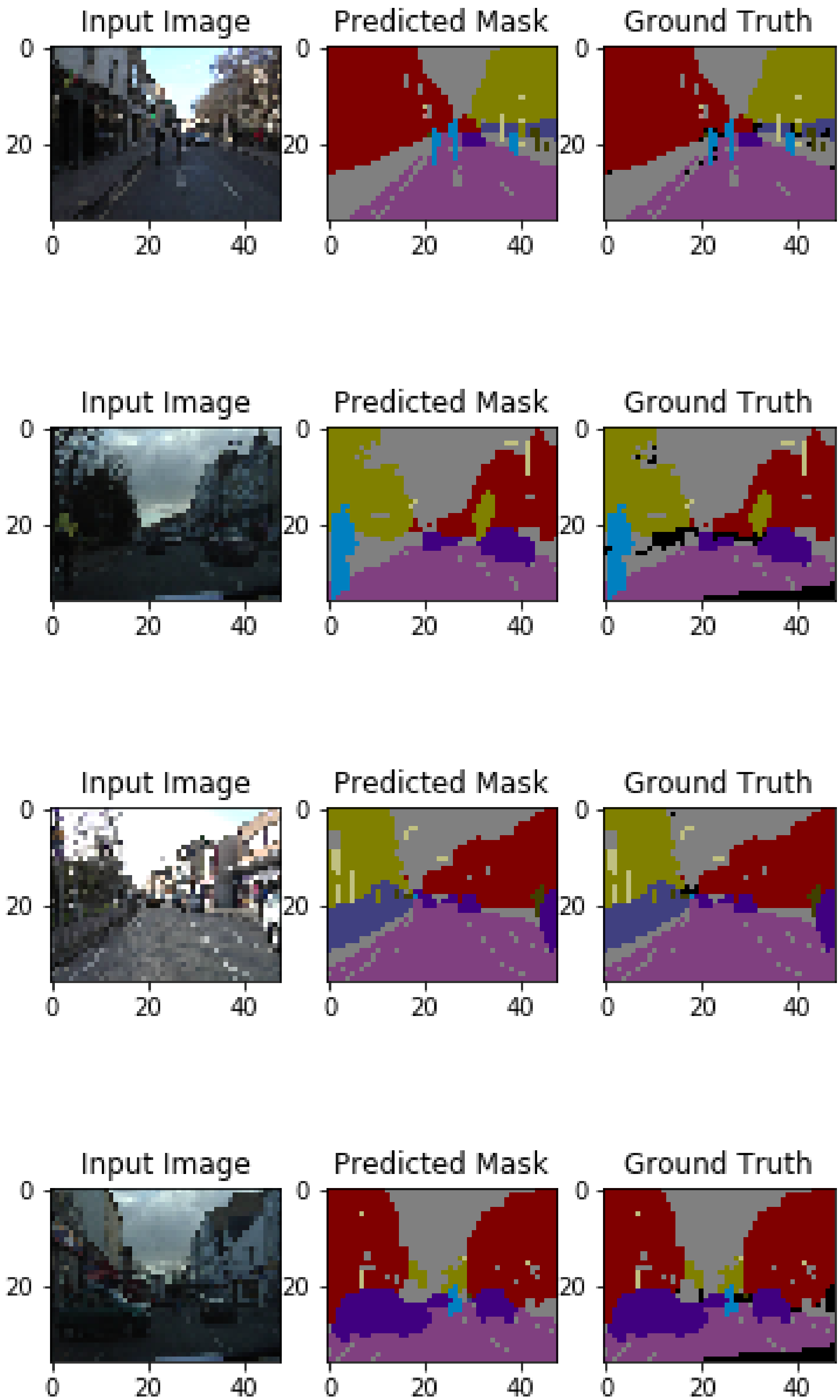
```
# Learning curve
x1 = range(0, NUM_EPOCHS)
y1 = acc_list
plt.figure
plt.plot(x1, y1, 'b-', label="train_acc")
plt.xlabel('Test accuracy/loss vs. epoches')
```

```
plt.ylabel('Test accuracy/loss')
x2 = range(0, NUM_EPOCHS)
y2 = loss_list
plt.plot(x2, y2, 'y-', label="train_loss")
plt.legend(loc="center right")
plt.show()
```



[27]:
```
# test

SAVED_MODEL_PATH = "./model_best.pth"
OUTPUT_DIR = "./result_image"

def validate():
    model.eval()

    input_tensor = torch.autograd.Variable(x)
    target_tensor = torch.autograd.Variable(y)

    if CUDA:
        input_tensor = input_tensor.cuda()
        target_tensor = target_tensor.cuda()

    predicted_tensor, softmaxed_tensor = model(input_tensor)
    loss = criterion(predicted_tensor, target_tensor)
```

```
    for idx, predicted_mask in enumerate(softmaxed_tensor):

        target_mask = target_tensor[idx]
        input_image = input_tensor[idx]
        fig = plt.figure()

        a = fig.add_subplot(1,3,1)
        plt.imshow(input_image.cpu().permute(1,2,0))
        a.set_title('Input Image')

        a = fig.add_subplot(1,3,2)
        predicted_mx = predicted_mask.detach().cpu().numpy()
        predicted_mx = predicted_mx.argmax(axis=0)
        predicted_rgb = label_to_rgb(predicted_mx, color_encoding)
        plt.imshow(predicted_rgb)
        a.set_title('Predicted Mask')

        a = fig.add_subplot(1,3,3)
        target_mx = target_mask.detach().cpu().numpy()
        target_rgb = label_to_rgb(target_mx, color_encoding)
        plt.imshow(target_rgb)
        a.set_title('Ground Truth')


if __name__ == "__main__":

    CUDA = torch.cuda.is_available()

    if CUDA:
        model = SegNet(input_channels=NUM_INPUT_CHANNELS,
                       output_channels=NUM_OUTPUT_CHANNELS).cuda()

        class_weights = torch.FloatTensor(CAMVID_CLASS_WEIGHTS).cuda()
        criterion = torch.nn.CrossEntropyLoss(weight=class_weights).cuda()
    else:
        model = SegNet(input_channels=NUM_INPUT_CHANNELS,
                       output_channels=NUM_OUTPUT_CHANNELS)

        class_weights = torch.FloatTensor(CAMVID_CLASS_WEIGHTS).cuda()
        criterion = torch.nn.CrossEntropyLoss(weight=class_weights)


    model.load_state_dict(torch.load(SAVED_MODEL_PATH))


    validate()
```


Input Image
Predicted Mask
Ground Truth
Input Image
Predicted Mask
Ground Truth
Input Image
Predicted Mask
Ground Truth
Input Image
Predicted Mask
Ground Truth

Test for whole dataset

[8]:
```
"""
Train a SegNet model
Usage:
python train.py --data_root /home/SharedData/intern_sayan/PascalVOC2012/data/
↪VOCdevkit/VOC2012/ \
                --train_path ImageSets/Segmentation/train.txt \
                --img_dir JPEGImages \
                --mask_dir SegmentationClass \
                --save_dir /home/SharedData/intern_sayan/PascalVOC2012/ \
                --checkpoint /home/SharedData/intern_sayan/PascalVOC2012/
↪model_best.pth \
                --gpu 1
"""

from __future__ import print_function
import os
import time
import torch
from torch.utils.data import DataLoader
from torch.autograd import Variable



# Parameters
NUM_INPUT_CHANNELS = 3
NUM_OUTPUT_CHANNELS = 13

NUM_EPOCHS = 160

LEARNING_RATE = 1e-3
MOMENTUM = 0.9
BATCH_SIZE = 4



def train():
    is_better = True
    prev_loss = float('inf')
#     model.load_state_dict(torch.load("./model_best.pth"))
    model.train()

    loss_list = []
    acc_list = []
    for epoch in range(NUM_EPOCHS):
        loss_f = 0
```

```
        t_start = time.time()

        for index,[img,label] in enumerate(train_dataloader):
            input_tensor = Variable(img)
            target_tensor = Variable(label)
            if CUDA:
                input_tensor = input_tensor.cuda()
                target_tensor = target_tensor.cuda()

            predicted_tensor, softmaxed_tensor = model(input_tensor)


            optimizer.zero_grad()
            loss = criterion(predicted_tensor, target_tensor)
            loss.backward()
            optimizer.step()


            loss_f += loss.float()


            delta = time.time() - t_start
            is_better = loss_f < prev_loss

            if is_better:
                prev_loss = loss_f
            if index%40== 0:
                print("Epoch #{}\tTotal loss: {:.8f}\t Time: {:2f}s".
↪format(epoch+1, loss_f , delta))
            if index == (len(train_dataloader) - 1):
                loss_list.append(loss_f)
                _, prediction = torch.max(predicted_tensor,1)
                y_true = torch.flatten(target_tensor)
                y_prep = torch.flatten(prediction)
                intersection = torch.sum(y_true * y_prep)
                acc = (2. * intersection) / (torch.sum(y_true*y_true) + torch.
↪sum(y_prep*y_prep))
                acc = acc.detach().cpu().numpy()
                acc_list.append(acc)
        torch.save(model.state_dict(), os.path.join('/content/drive/My Drive/
↪segnet', "model_best.pth"))
    return loss_list,acc_list

if __name__ == "__main__":


    CUDA = torch.cuda.is_available()
```

```python
# weights when using median frequency balancing used in SegNet paper
# https://arxiv.org/pdf/1511.00561.pdf
# The numbers were generated by:
# https://github.com/yandex/segnet-torch/blob/master/datasets/camvid-gen.lua
CAMVID_CLASS_WEIGHTS = [0.58872014284134,
             0.51052379608154,
             2.6966278553009,
             0.45021694898605,
             1.1785038709641,
             0.77028578519821,
             2.4782588481903,
             2.5273461341858,
             1.0122526884079,
             3.2375309467316,
             4.1312313079834,
             0.3,
             0]
if CUDA:
    model = SegNet(input_channels=NUM_INPUT_CHANNELS,␣
↪output_channels=NUM_OUTPUT_CHANNELS).cuda()
    class_weights = torch.FloatTensor(CAMVID_CLASS_WEIGHTS).cuda()
    criterion = torch.nn.CrossEntropyLoss(weight=class_weights).cuda()
else:
    model = SegNet(input_channels=NUM_INPUT_CHANNELS,␣
↪output_channels=NUM_OUTPUT_CHANNELS)
    class_weights = torch.FloatTensor(CAMVID_CLASS_WEIGHTS)
    criterion = torch.nn.CrossEntropyLoss(weight=class_weights)


optimizer = torch.optim.Adam(model.parameters(), lr=LEARNING_RATE)

loss_list = []
acc_list = []
loss_list, acc_list= train()
```

```
Epoch #1        Total loss: 2.79073524   Time: 3.158261s
Epoch #1        Total loss: 52.70711517  Time: 120.173725s
Epoch #1        Total loss: 88.79529572  Time: 225.196719s
Epoch #2        Total loss: 0.89505726   Time: 0.196335s
Epoch #2        Total loss: 34.81172180  Time: 8.891248s
Epoch #2        Total loss: 67.23095703  Time: 17.123669s
Epoch #3        Total loss: 0.66244602   Time: 0.206405s
Epoch #3        Total loss: 30.05274200  Time: 7.692185s
Epoch #3        Total loss: 57.62046814  Time: 15.973802s
Epoch #4        Total loss: 0.74270427   Time: 0.294773s
```

```
Epoch #4        Total loss: 28.73457336  Time: 7.664445s
Epoch #4        Total loss: 55.50716400  Time: 15.843583s
Epoch #5        Total loss: 0.51564145   Time: 0.194166s
Epoch #5        Total loss: 25.89620972  Time: 7.567474s
Epoch #5        Total loss: 51.41121674  Time: 15.462481s
Epoch #6        Total loss: 0.64950240   Time: 0.193268s
Epoch #6        Total loss: 27.27328300  Time: 7.767581s
Epoch #6        Total loss: 54.92087936  Time: 15.842180s
Epoch #7        Total loss: 0.48998925   Time: 0.186098s
Epoch #7        Total loss: 26.42895889  Time: 7.957027s
Epoch #7        Total loss: 52.58362579  Time: 16.471228s
Epoch #8        Total loss: 0.74209774   Time: 0.223622s
Epoch #8        Total loss: 27.88312340  Time: 7.908955s
Epoch #8        Total loss: 53.17484665  Time: 16.348693s
Epoch #9        Total loss: 0.53831977   Time: 0.201347s
Epoch #9        Total loss: 24.60177231  Time: 7.974913s
Epoch #9        Total loss: 49.43677139  Time: 16.215892s
Epoch #10       Total loss: 0.72697365   Time: 0.412176s
Epoch #10       Total loss: 28.23324776  Time: 7.981323s
Epoch #10       Total loss: 53.09757996  Time: 16.176852s
Epoch #11       Total loss: 0.68504590   Time: 0.200847s
Epoch #11       Total loss: 24.53111076  Time: 7.780811s
Epoch #11       Total loss: 49.10469437  Time: 16.369132s
Epoch #12       Total loss: 0.61855805   Time: 0.407448s
Epoch #12       Total loss: 22.39270401  Time: 8.153446s
Epoch #12       Total loss: 44.23672104  Time: 16.618912s
Epoch #13       Total loss: 0.42640942   Time: 0.374184s
Epoch #13       Total loss: 22.01772499  Time: 8.056992s
Epoch #13       Total loss: 44.43247986  Time: 16.463574s
Epoch #14       Total loss: 0.46345830   Time: 0.465798s
Epoch #14       Total loss: 20.59221840  Time: 8.092975s
Epoch #14       Total loss: 39.98138809  Time: 16.474354s
Epoch #15       Total loss: 0.59995335   Time: 0.205059s
Epoch #15       Total loss: 19.70584297  Time: 7.886991s
Epoch #15       Total loss: 38.87735748  Time: 16.212556s
Epoch #16       Total loss: 0.49734887   Time: 0.245718s
Epoch #16       Total loss: 21.05565643  Time: 7.759240s
Epoch #16       Total loss: 40.55717850  Time: 15.930613s
Epoch #17       Total loss: 0.42771584   Time: 0.251478s
Epoch #17       Total loss: 18.42227745  Time: 7.792841s
Epoch #17       Total loss: 37.36229324  Time: 15.811461s
Epoch #18       Total loss: 0.45413458   Time: 0.204554s
Epoch #18       Total loss: 17.72665977  Time: 7.821945s
Epoch #18       Total loss: 36.39314270  Time: 16.289325s
Epoch #19       Total loss: 0.33608621   Time: 0.200009s
Epoch #19       Total loss: 17.17278099  Time: 7.769358s
Epoch #19       Total loss: 34.61000061  Time: 15.870885s
Epoch #20       Total loss: 0.54773474   Time: 0.809238s
```

```
Epoch #20       Total loss: 16.87650681  Time: 8.347552s
Epoch #20       Total loss: 34.31000137  Time: 16.543852s
Epoch #21       Total loss: 0.39975718   Time: 0.192823s
Epoch #21       Total loss: 17.36980247  Time: 7.503309s
Epoch #21       Total loss: 33.26158524  Time: 15.483444s
Epoch #22       Total loss: 0.43039480   Time: 0.183169s
Epoch #22       Total loss: 17.80486107  Time: 7.606499s
Epoch #22       Total loss: 33.52763748  Time: 15.504840s
Epoch #23       Total loss: 0.38034090   Time: 0.206482s
Epoch #23       Total loss: 15.01726723  Time: 7.570601s
Epoch #23       Total loss: 30.50909233  Time: 15.484264s
Epoch #24       Total loss: 0.35522732   Time: 0.176549s
Epoch #24       Total loss: 15.95325756  Time: 7.603338s
Epoch #24       Total loss: 30.76111984  Time: 15.536719s
Epoch #25       Total loss: 0.50881809   Time: 0.186021s
Epoch #25       Total loss: 13.93685150  Time: 7.468513s
Epoch #25       Total loss: 28.21644211  Time: 15.367339s
Epoch #26       Total loss: 0.33947930   Time: 0.236910s
Epoch #26       Total loss: 14.77280617  Time: 7.518933s
Epoch #26       Total loss: 30.83484268  Time: 15.434861s
Epoch #27       Total loss: 0.30337632   Time: 0.180359s
Epoch #27       Total loss: 15.20313644  Time: 7.381647s
Epoch #27       Total loss: 30.13105011  Time: 15.254979s
Epoch #28       Total loss: 0.30573216   Time: 0.383163s
Epoch #28       Total loss: 14.43111229  Time: 7.611681s
Epoch #28       Total loss: 27.69159317  Time: 15.530839s
Epoch #29       Total loss: 0.42956463   Time: 0.178907s
Epoch #29       Total loss: 16.00214386  Time: 7.504362s
Epoch #29       Total loss: 35.28704834  Time: 15.438761s
Epoch #30       Total loss: 0.40119320   Time: 0.196252s
Epoch #30       Total loss: 16.87842751  Time: 7.429649s
Epoch #30       Total loss: 34.23440552  Time: 15.355723s
Epoch #31       Total loss: 0.36637092   Time: 0.175045s
Epoch #31       Total loss: 15.41750717  Time: 7.447023s
Epoch #31       Total loss: 29.84320831  Time: 15.270514s
Epoch #32       Total loss: 0.37262589   Time: 0.251554s
Epoch #32       Total loss: 18.47575951  Time: 7.463608s
Epoch #32       Total loss: 35.51514435  Time: 15.385399s
Epoch #33       Total loss: 0.48846874   Time: 0.181816s
Epoch #33       Total loss: 15.25479889  Time: 7.427532s
Epoch #33       Total loss: 29.82523537  Time: 15.336196s
Epoch #34       Total loss: 0.34673735   Time: 0.567045s
Epoch #34       Total loss: 15.30326080  Time: 7.707755s
Epoch #34       Total loss: 29.16212463  Time: 15.552392s
Epoch #35       Total loss: 0.35141057   Time: 0.176940s
Epoch #35       Total loss: 12.55681705  Time: 7.372004s
Epoch #35       Total loss: 24.52288818  Time: 15.237844s
Epoch #36       Total loss: 0.35156521   Time: 0.186207s
```

```
Epoch #36       Total loss: 12.37071228  Time: 7.451581s
Epoch #36       Total loss: 24.92478943  Time: 15.299366s
Epoch #37       Total loss: 0.24568884   Time: 0.159665s
Epoch #37       Total loss: 11.10037899  Time: 7.310243s
Epoch #37       Total loss: 21.78655052  Time: 15.403590s
Epoch #38       Total loss: 0.32508337   Time: 0.410890s
Epoch #38       Total loss: 10.96766853  Time: 7.559850s
Epoch #38       Total loss: 22.72075653  Time: 15.405246s
Epoch #39       Total loss: 0.47703439   Time: 0.214341s
Epoch #39       Total loss: 15.94780636  Time: 7.354910s
Epoch #39       Total loss: 29.26250839  Time: 15.193760s
Epoch #40       Total loss: 0.22967039   Time: 0.315267s
Epoch #40       Total loss: 12.00907803  Time: 7.464911s
Epoch #40       Total loss: 23.06777954  Time: 15.131840s
Epoch #41       Total loss: 0.25352502   Time: 0.290790s
Epoch #41       Total loss: 10.80625916  Time: 7.428892s
Epoch #41       Total loss: 21.83430672  Time: 15.281844s
Epoch #42       Total loss: 0.21151462   Time: 0.209105s
Epoch #42       Total loss: 9.63568592   Time: 7.386911s
Epoch #42       Total loss: 20.20479012  Time: 15.234480s
Epoch #43       Total loss: 0.20672701   Time: 0.194951s
Epoch #43       Total loss: 10.16579914  Time: 7.435892s
Epoch #43       Total loss: 20.61584091  Time: 15.224571s
Epoch #44       Total loss: 0.38105777   Time: 0.207663s
Epoch #44       Total loss: 13.04780197  Time: 7.379904s
Epoch #44       Total loss: 24.87861252  Time: 15.302292s
Epoch #45       Total loss: 0.30065984   Time: 0.357611s
Epoch #45       Total loss: 11.59985924  Time: 7.557507s
Epoch #45       Total loss: 23.56425095  Time: 15.311113s
Epoch #46       Total loss: 0.16784479   Time: 0.302363s
Epoch #46       Total loss: 11.22827148  Time: 7.508335s
Epoch #46       Total loss: 22.39805984  Time: 15.342592s
Epoch #47       Total loss: 0.24182041   Time: 0.225774s
Epoch #47       Total loss: 9.81881714   Time: 7.416037s
Epoch #47       Total loss: 18.77026749  Time: 15.102608s
Epoch #48       Total loss: 0.17978711   Time: 0.256214s
Epoch #48       Total loss: 8.79106426   Time: 7.461378s
Epoch #48       Total loss: 17.33232117  Time: 15.292384s
Epoch #49       Total loss: 0.22315247   Time: 0.274798s
Epoch #49       Total loss: 8.21559811   Time: 7.547099s
Epoch #49       Total loss: 16.21633720  Time: 15.336759s
Epoch #50       Total loss: 0.27824315   Time: 0.217484s
Epoch #50       Total loss: 8.04436970   Time: 7.410898s
Epoch #50       Total loss: 15.86022568  Time: 15.185818s
Epoch #51       Total loss: 0.16689165   Time: 0.399042s
Epoch #51       Total loss: 7.41523933   Time: 7.616218s
Epoch #51       Total loss: 14.46589088  Time: 15.448824s
Epoch #52       Total loss: 0.13586032   Time: 0.246193s
```

```
Epoch #52       Total loss: 7.54118919   Time: 7.394165s
Epoch #52       Total loss: 15.09787941  Time: 15.238204s
Epoch #53       Total loss: 0.25433078   Time: 0.307779s
Epoch #53       Total loss: 7.55999756   Time: 7.478351s
Epoch #53       Total loss: 15.45461273  Time: 15.222769s
Epoch #54       Total loss: 0.20231220   Time: 0.205498s
Epoch #54       Total loss: 7.75809097   Time: 7.371165s
Epoch #54       Total loss: 15.36217403  Time: 15.131016s
Epoch #55       Total loss: 0.18836553   Time: 0.163409s
Epoch #55       Total loss: 8.41700840   Time: 7.297954s
Epoch #55       Total loss: 16.26173019  Time: 15.169752s
Epoch #56       Total loss: 0.16360240   Time: 0.177052s
Epoch #56       Total loss: 6.98355722   Time: 7.357804s
Epoch #56       Total loss: 13.88643646  Time: 15.124029s
Epoch #57       Total loss: 0.18606143   Time: 0.182432s
Epoch #57       Total loss: 6.43459654   Time: 7.376367s
Epoch #57       Total loss: 13.31478500  Time: 15.269836s
Epoch #58       Total loss: 0.11146921   Time: 0.186228s
Epoch #58       Total loss: 6.88897562   Time: 7.425875s
Epoch #58       Total loss: 13.26956081  Time: 15.198689s
Epoch #59       Total loss: 0.13871455   Time: 0.184455s
Epoch #59       Total loss: 6.21291351   Time: 7.405334s
Epoch #59       Total loss: 12.23571110  Time: 15.188972s
Epoch #60       Total loss: 0.14386167   Time: 0.450551s
Epoch #60       Total loss: 5.64130449   Time: 7.629508s
Epoch #60       Total loss: 11.18725586  Time: 15.504489s
Epoch #61       Total loss: 0.10836481   Time: 0.173029s
Epoch #61       Total loss: 7.52743864   Time: 7.313670s
Epoch #61       Total loss: 17.30227661  Time: 15.135767s
Epoch #62       Total loss: 0.26762527   Time: 0.181603s
Epoch #62       Total loss: 22.52822495  Time: 7.273490s
Epoch #62       Total loss: 43.45865250  Time: 14.953768s
Epoch #63       Total loss: 0.51760143   Time: 0.171182s
Epoch #63       Total loss: 17.58595657  Time: 7.349326s
Epoch #63       Total loss: 33.94009018  Time: 15.092829s
Epoch #64       Total loss: 0.35991395   Time: 0.178358s
Epoch #64       Total loss: 11.49263859  Time: 7.540679s
Epoch #64       Total loss: 23.05109978  Time: 15.431317s
Epoch #65       Total loss: 0.21813080   Time: 0.172369s
Epoch #65       Total loss: 9.47453499   Time: 7.271712s
Epoch #65       Total loss: 18.58236313  Time: 14.933399s
Epoch #66       Total loss: 0.16488995   Time: 0.176511s
Epoch #66       Total loss: 8.44415474   Time: 7.289031s
Epoch #66       Total loss: 16.52661133  Time: 14.936495s
Epoch #67       Total loss: 0.32048601   Time: 0.164366s
Epoch #67       Total loss: 7.87690687   Time: 7.600685s
Epoch #67       Total loss: 15.52514362  Time: 15.897325s
Epoch #68       Total loss: 0.17968494   Time: 0.201982s
```

```
Epoch #68       Total loss: 7.06397963   Time: 7.829511s
Epoch #68       Total loss: 14.39227962  Time: 16.232228s
Epoch #69       Total loss: 0.12908733   Time: 0.200681s
Epoch #69       Total loss: 6.91664743   Time: 7.809687s
Epoch #69       Total loss: 13.49976921  Time: 16.162857s
Epoch #70       Total loss: 0.14612368   Time: 0.186625s
Epoch #70       Total loss: 6.34257507   Time: 7.838933s
Epoch #70       Total loss: 12.30103397  Time: 16.361441s
Epoch #71       Total loss: 0.12584981   Time: 0.199649s
Epoch #71       Total loss: 5.97572517   Time: 7.818001s
Epoch #71       Total loss: 11.50465107  Time: 16.120608s
Epoch #72       Total loss: 0.13791458   Time: 0.186275s
Epoch #72       Total loss: 6.45688009   Time: 7.766879s
Epoch #72       Total loss: 13.68082523  Time: 15.913530s
Epoch #73       Total loss: 0.12559366   Time: 0.184407s
Epoch #73       Total loss: 5.91043234   Time: 7.788450s
Epoch #73       Total loss: 11.82211971  Time: 16.118456s
Epoch #74       Total loss: 0.16804776   Time: 0.193503s
Epoch #74       Total loss: 6.15995646   Time: 7.837401s
Epoch #74       Total loss: 11.69609928  Time: 16.120982s
Epoch #75       Total loss: 0.11361320   Time: 0.178457s
Epoch #75       Total loss: 5.55689192   Time: 7.877071s
Epoch #75       Total loss: 10.67031193  Time: 16.120674s
Epoch #76       Total loss: 0.14774768   Time: 0.191190s
Epoch #76       Total loss: 5.29548931   Time: 7.972161s
Epoch #76       Total loss: 10.43679619  Time: 16.220072s
Epoch #77       Total loss: 0.11461330   Time: 0.206772s
Epoch #77       Total loss: 5.49796677   Time: 8.113926s
Epoch #77       Total loss: 10.92245960  Time: 16.437419s
Epoch #78       Total loss: 0.14723153   Time: 0.187198s
Epoch #78       Total loss: 5.60879469   Time: 7.996922s
Epoch #78       Total loss: 11.59295177  Time: 16.173334s
Epoch #79       Total loss: 0.12484452   Time: 0.190699s
Epoch #79       Total loss: 5.53596020   Time: 7.811692s
Epoch #79       Total loss: 11.24941635  Time: 16.435960s
Epoch #80       Total loss: 0.13350672   Time: 0.206242s
Epoch #80       Total loss: 5.09477234   Time: 8.274406s
Epoch #80       Total loss: 10.53414726  Time: 16.941365s
Epoch #81       Total loss: 0.13318625   Time: 0.998603s
Epoch #81       Total loss: 4.80633163   Time: 9.221313s
Epoch #81       Total loss: 9.81284904   Time: 18.052896s
Epoch #82       Total loss: 0.13543141   Time: 0.214278s
Epoch #82       Total loss: 5.03738070   Time: 8.206817s
Epoch #82       Total loss: 10.10075855  Time: 17.022867s
Epoch #83       Total loss: 0.12292805   Time: 0.190571s
Epoch #83       Total loss: 5.05745935   Time: 8.407451s
Epoch #83       Total loss: 10.05570412  Time: 17.432245s
Epoch #84       Total loss: 0.13884325   Time: 0.213753s
```

```
Epoch #84        Total loss: 4.78088951   Time: 8.823522s
Epoch #84        Total loss: 9.61703491   Time: 17.890411s
Epoch #85        Total loss: 0.14235201   Time: 0.216515s
Epoch #85        Total loss: 5.50257206   Time: 8.216386s
Epoch #85        Total loss: 11.13757229  Time: 16.861474s
Epoch #86        Total loss: 0.17665485   Time: 0.198692s
Epoch #86        Total loss: 5.60017729   Time: 8.278206s
Epoch #86        Total loss: 10.95112419  Time: 16.779225s
Epoch #87        Total loss: 0.11005562   Time: 0.575612s
Epoch #87        Total loss: 5.21832132   Time: 8.257239s
Epoch #87        Total loss: 10.09813404  Time: 16.660527s
Epoch #88        Total loss: 0.12479895   Time: 0.190830s
Epoch #88        Total loss: 4.72096825   Time: 7.934304s
Epoch #88        Total loss: 9.61163616   Time: 16.254495s
Epoch #89        Total loss: 0.11709055   Time: 0.219411s
Epoch #89        Total loss: 4.77782297   Time: 8.265243s
Epoch #89        Total loss: 9.21133041   Time: 16.867628s
Epoch #90        Total loss: 0.12548842   Time: 0.197069s
Epoch #90        Total loss: 4.33230782   Time: 8.027630s
Epoch #90        Total loss: 9.22784138   Time: 16.446584s
Epoch #91        Total loss: 0.16294156   Time: 0.188218s
Epoch #91        Total loss: 5.37274742   Time: 8.122585s
Epoch #91        Total loss: 14.91723156  Time: 16.648874s
Epoch #92        Total loss: 0.17041677   Time: 0.203429s
Epoch #92        Total loss: 10.00752544  Time: 8.107420s
Epoch #92        Total loss: 19.35751343  Time: 16.313532s
Epoch #93        Total loss: 0.18508707   Time: 0.200658s
Epoch #93        Total loss: 7.72218990   Time: 8.009469s
Epoch #93        Total loss: 14.94191647  Time: 16.457192s
Epoch #94        Total loss: 0.16343530   Time: 0.215199s
Epoch #94        Total loss: 6.72229052   Time: 7.919339s
Epoch #94        Total loss: 12.45981407  Time: 16.503967s
Epoch #95        Total loss: 0.11938729   Time: 0.179026s
Epoch #95        Total loss: 5.25648975   Time: 7.856662s
Epoch #95        Total loss: 10.56264210  Time: 16.176284s
Epoch #96        Total loss: 0.09528979   Time: 0.201320s
Epoch #96        Total loss: 4.82281065   Time: 7.894253s
Epoch #96        Total loss: 9.55266285   Time: 16.367084s
Epoch #97        Total loss: 0.09730206   Time: 0.205249s
Epoch #97        Total loss: 4.46385193   Time: 8.077306s
Epoch #97        Total loss: 8.76303101   Time: 16.444807s
Epoch #98        Total loss: 0.11357366   Time: 0.206909s
Epoch #98        Total loss: 4.25098944   Time: 7.919779s
Epoch #98        Total loss: 8.37553883   Time: 16.390149s
Epoch #99        Total loss: 0.08575506   Time: 0.200410s
Epoch #99        Total loss: 4.39531326   Time: 7.763487s
Epoch #99        Total loss: 8.36770916   Time: 16.033583s
Epoch #100       Total loss: 0.11028317   Time: 0.195043s
```

```
Epoch #100      Total loss: 4.01414824   Time: 7.716946s
Epoch #100      Total loss: 7.89959192   Time: 15.818573s
Epoch #101      Total loss: 0.06925537   Time: 0.209399s
Epoch #101      Total loss: 4.05975199   Time: 8.210423s
Epoch #101      Total loss: 8.19833374   Time: 16.761515s
Epoch #102      Total loss: 0.11370429   Time: 0.180517s
Epoch #102      Total loss: 3.84588909   Time: 7.970098s
Epoch #102      Total loss: 7.72584820   Time: 16.293585s
Epoch #103      Total loss: 0.09912010   Time: 0.219903s
Epoch #103      Total loss: 3.71868277   Time: 7.673444s
Epoch #103      Total loss: 7.46599388   Time: 15.969617s
Epoch #104      Total loss: 0.09330463   Time: 0.191726s
Epoch #104      Total loss: 4.83966923   Time: 7.628542s
Epoch #104      Total loss: 9.68351936   Time: 15.529906s
Epoch #105      Total loss: 0.10503995   Time: 0.190225s
Epoch #105      Total loss: 4.38699436   Time: 7.598211s
Epoch #105      Total loss: 8.68557167   Time: 15.764383s
Epoch #106      Total loss: 0.10274321   Time: 0.192111s
Epoch #106      Total loss: 4.19416666   Time: 7.730830s
Epoch #106      Total loss: 8.40557194   Time: 15.737056s
Epoch #107      Total loss: 0.12908451   Time: 0.204476s
Epoch #107      Total loss: 4.17168570   Time: 7.658205s
Epoch #107      Total loss: 8.10866928   Time: 15.749087s
Epoch #108      Total loss: 0.09755995   Time: 0.181942s
Epoch #108      Total loss: 3.94235444   Time: 7.564597s
Epoch #108      Total loss: 8.00729656   Time: 15.582460s
Epoch #109      Total loss: 0.11044141   Time: 0.190091s
Epoch #109      Total loss: 13.40043068  Time: 7.675172s
Epoch #109      Total loss: 31.28546715  Time: 15.494831s
Epoch #110      Total loss: 0.32479265   Time: 0.182283s
Epoch #110      Total loss: 12.57219315  Time: 7.551666s
Epoch #110      Total loss: 21.88905716  Time: 15.514877s
Epoch #111      Total loss: 0.22977939   Time: 0.175295s
Epoch #111      Total loss: 7.57708549   Time: 7.549559s
Epoch #111      Total loss: 15.15770435  Time: 15.599338s
Epoch #112      Total loss: 0.17266135   Time: 0.202516s
Epoch #112      Total loss: 6.02539110   Time: 7.554028s
Epoch #112      Total loss: 11.30610657  Time: 15.721478s
Epoch #113      Total loss: 0.13099469   Time: 0.188635s
Epoch #113      Total loss: 4.63828754   Time: 7.729051s
Epoch #113      Total loss: 9.46577930   Time: 15.921593s
Epoch #114      Total loss: 0.10014349   Time: 0.195898s
Epoch #114      Total loss: 4.33158636   Time: 7.892416s
Epoch #114      Total loss: 8.74739075   Time: 15.976326s
Epoch #115      Total loss: 0.09349649   Time: 0.182276s
Epoch #115      Total loss: 4.27468443   Time: 7.671766s
Epoch #115      Total loss: 8.18968582   Time: 15.806005s
Epoch #116      Total loss: 0.07677813   Time: 0.181106s
```

```
Epoch #116      Total loss: 4.05060005   Time: 7.427478s
Epoch #116      Total loss: 7.99387598   Time: 15.386001s
Epoch #117      Total loss: 0.07024020   Time: 0.189835s
Epoch #117      Total loss: 3.93690133   Time: 7.465838s
Epoch #117      Total loss: 7.82861185   Time: 15.244781s
Epoch #118      Total loss: 0.09780766   Time: 0.186970s
Epoch #118      Total loss: 3.76240015   Time: 7.478864s
Epoch #118      Total loss: 7.38265753   Time: 15.531018s
Epoch #119      Total loss: 0.08668015   Time: 0.192884s
Epoch #119      Total loss: 3.64459348   Time: 7.914633s
Epoch #119      Total loss: 7.21307611   Time: 15.758930s
Epoch #120      Total loss: 0.08306948   Time: 0.193478s
Epoch #120      Total loss: 3.66238999   Time: 7.587528s
Epoch #120      Total loss: 7.50117874   Time: 15.780506s
Epoch #121      Total loss: 0.07867153   Time: 0.184141s
Epoch #121      Total loss: 3.62266803   Time: 7.415211s
Epoch #121      Total loss: 7.35357761   Time: 15.166422s
Epoch #122      Total loss: 0.08435074   Time: 0.184600s
Epoch #122      Total loss: 3.52651024   Time: 7.412456s
Epoch #122      Total loss: 7.11246109   Time: 15.218717s
Epoch #123      Total loss: 0.06576300   Time: 0.185416s
Epoch #123      Total loss: 3.45994210   Time: 7.361278s
Epoch #123      Total loss: 6.95947599   Time: 15.138936s
Epoch #124      Total loss: 0.06928811   Time: 0.178518s
Epoch #124      Total loss: 3.28147244   Time: 7.470210s
Epoch #124      Total loss: 6.58287954   Time: 15.293926s
Epoch #125      Total loss: 0.09120315   Time: 0.169430s
Epoch #125      Total loss: 3.29278970   Time: 7.349483s
Epoch #125      Total loss: 6.51277781   Time: 14.893424s
Epoch #126      Total loss: 0.05617649   Time: 0.180432s
Epoch #126      Total loss: 3.29943252   Time: 7.376539s
Epoch #126      Total loss: 6.45654058   Time: 15.133730s
Epoch #127      Total loss: 0.06876162   Time: 0.180581s
Epoch #127      Total loss: 3.05317283   Time: 7.429797s
Epoch #127      Total loss: 6.13008213   Time: 15.427504s
Epoch #128      Total loss: 0.08044403   Time: 0.171240s
Epoch #128      Total loss: 3.49249458   Time: 7.275644s
Epoch #128      Total loss: 6.70160627   Time: 15.028857s
Epoch #129      Total loss: 0.10716099   Time: 0.185010s
Epoch #129      Total loss: 3.78790641   Time: 7.410002s
Epoch #129      Total loss: 7.60303354   Time: 15.212306s
Epoch #130      Total loss: 0.06745390   Time: 0.175373s
Epoch #130      Total loss: 3.69922352   Time: 7.325103s
Epoch #130      Total loss: 7.28573370   Time: 15.042457s
Epoch #131      Total loss: 0.08976488   Time: 0.181720s
Epoch #131      Total loss: 3.65254831   Time: 7.398531s
Epoch #131      Total loss: 7.50702429   Time: 15.243118s
Epoch #132      Total loss: 0.10312302   Time: 0.177512s
```

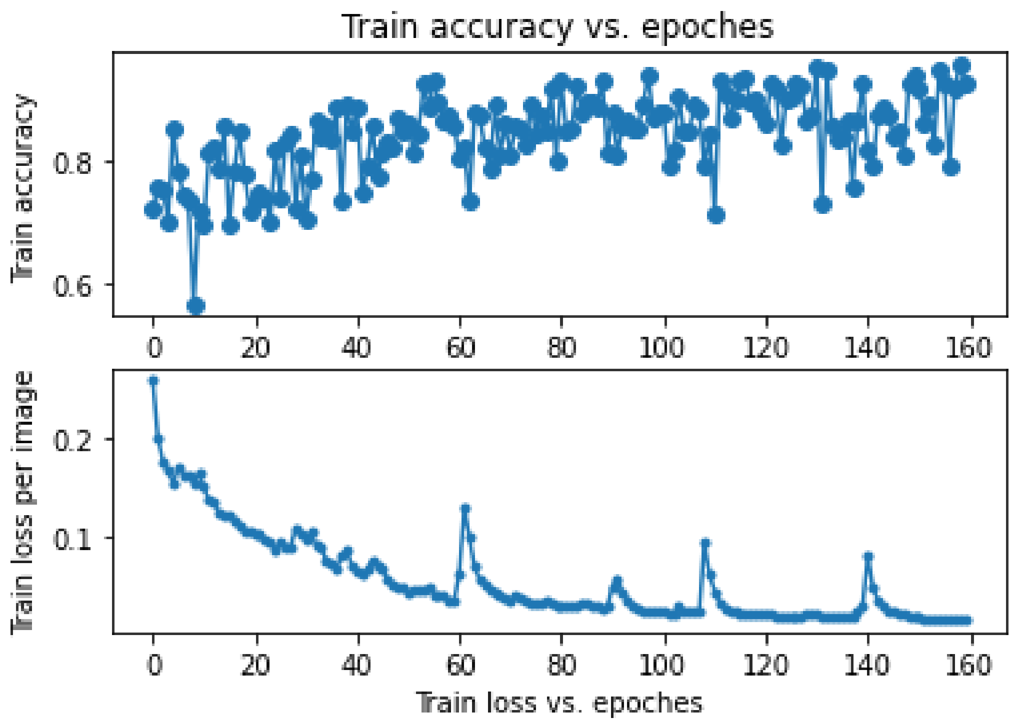
```
Epoch #132      Total loss: 3.44609499   Time: 7.300535s
Epoch #132      Total loss: 6.78670454   Time: 14.981159s
Epoch #133      Total loss: 0.08445737   Time: 0.170148s
Epoch #133      Total loss: 3.23099113   Time: 7.400976s
Epoch #133      Total loss: 6.52785683   Time: 15.889786s
Epoch #134      Total loss: 0.08857644   Time: 0.200073s
Epoch #134      Total loss: 3.26301575   Time: 8.091557s
Epoch #134      Total loss: 6.41276503   Time: 16.651599s
Epoch #135      Total loss: 0.06457938   Time: 0.184986s
Epoch #135      Total loss: 3.01787257   Time: 7.895486s
Epoch #135      Total loss: 6.23055172   Time: 15.520697s
Epoch #136      Total loss: 0.07028575   Time: 0.190239s
Epoch #136      Total loss: 3.05905724   Time: 8.112459s
Epoch #136      Total loss: 6.17266941   Time: 16.159835s
Epoch #137      Total loss: 0.09520821   Time: 0.207305s
Epoch #137      Total loss: 3.14060783   Time: 8.048711s
Epoch #137      Total loss: 6.32169914   Time: 15.848579s
Epoch #138      Total loss: 0.05729937   Time: 0.209660s
Epoch #138      Total loss: 3.08809304   Time: 8.531389s
Epoch #138      Total loss: 6.26960516   Time: 16.431709s
Epoch #139      Total loss: 0.06828906   Time: 0.191434s
Epoch #139      Total loss: 3.36435676   Time: 7.942687s
Epoch #139      Total loss: 7.57109404   Time: 16.241220s
Epoch #140      Total loss: 0.09568826   Time: 0.223541s
Epoch #140      Total loss: 4.56288767   Time: 8.507626s
Epoch #140      Total loss: 9.23435211   Time: 16.422458s
Epoch #141      Total loss: 0.20904543   Time: 0.224469s
Epoch #141      Total loss: 10.90330505  Time: 8.068180s
Epoch #141      Total loss: 26.55052757  Time: 16.161376s
Epoch #142      Total loss: 0.19079681   Time: 0.204668s
Epoch #142      Total loss: 8.78553867   Time: 8.109647s
Epoch #142      Total loss: 16.50287247  Time: 16.205364s
Epoch #143      Total loss: 0.15287465   Time: 0.214536s
Epoch #143      Total loss: 6.20199633   Time: 8.100671s
Epoch #143      Total loss: 12.17472649  Time: 15.937210s
Epoch #144      Total loss: 0.09948855   Time: 0.212908s
Epoch #144      Total loss: 5.01955605   Time: 8.430067s
Epoch #144      Total loss: 10.25812244  Time: 16.367984s
Epoch #145      Total loss: 0.09879492   Time: 0.230086s
Epoch #145      Total loss: 4.49010515   Time: 8.426300s
Epoch #145      Total loss: 8.64378643   Time: 17.428938s
Epoch #146      Total loss: 0.09164838   Time: 0.207455s
Epoch #146      Total loss: 4.11086273   Time: 8.292027s
Epoch #146      Total loss: 7.91448164   Time: 16.366685s
Epoch #147      Total loss: 0.10102090   Time: 0.219768s
Epoch #147      Total loss: 3.65038824   Time: 8.104664s
Epoch #147      Total loss: 7.40128326   Time: 15.909150s
Epoch #148      Total loss: 0.09974888   Time: 0.205407s
```

```
Epoch #148      Total loss: 3.67055154   Time: 7.943307s
Epoch #148      Total loss: 7.25462818   Time: 15.619517s
Epoch #149      Total loss: 0.08021094   Time: 0.206821s
Epoch #149      Total loss: 3.44124651   Time: 8.006305s
Epoch #149      Total loss: 6.88147926   Time: 15.660192s
Epoch #150      Total loss: 0.07705156   Time: 0.202717s
Epoch #150      Total loss: 3.26406097   Time: 7.864746s
Epoch #150      Total loss: 6.40975380   Time: 15.561658s
Epoch #151      Total loss: 0.11058713   Time: 0.214940s
Epoch #151      Total loss: 3.13103604   Time: 8.233959s
Epoch #151      Total loss: 6.24022961   Time: 16.837000s
Epoch #152      Total loss: 0.06229006   Time: 0.207310s
Epoch #152      Total loss: 2.95474696   Time: 8.366466s
Epoch #152      Total loss: 6.02444601   Time: 16.672201s
Epoch #153      Total loss: 0.07468464   Time: 0.218290s
Epoch #153      Total loss: 2.87955213   Time: 8.156497s
Epoch #153      Total loss: 5.88207150   Time: 15.927642s
Epoch #154      Total loss: 0.08137368   Time: 0.213184s
Epoch #154      Total loss: 3.07300568   Time: 8.329954s
Epoch #154      Total loss: 6.02743101   Time: 16.273614s
Epoch #155      Total loss: 0.11463949   Time: 0.204260s
Epoch #155      Total loss: 2.94165254   Time: 7.978554s
Epoch #155      Total loss: 5.98151016   Time: 15.910996s
Epoch #156      Total loss: 0.06781179   Time: 0.201692s
Epoch #156      Total loss: 2.92896199   Time: 8.231076s
Epoch #156      Total loss: 5.76968288   Time: 16.321428s
Epoch #157      Total loss: 0.06371322   Time: 0.210765s
Epoch #157      Total loss: 2.88053584   Time: 8.138907s
Epoch #157      Total loss: 5.71014929   Time: 17.127388s
Epoch #158      Total loss: 0.07307822   Time: 0.210353s
Epoch #158      Total loss: 2.87134433   Time: 7.986340s
Epoch #158      Total loss: 5.83848810   Time: 15.723507s
Epoch #159      Total loss: 0.08712765   Time: 0.212622s
Epoch #159      Total loss: 2.87067533   Time: 8.170836s
Epoch #159      Total loss: 5.73263741   Time: 16.266184s
Epoch #160      Total loss: 0.08165640   Time: 0.201157s
Epoch #160      Total loss: 2.77239561   Time: 8.234527s
Epoch #160      Total loss: 5.40727901   Time: 16.042111s
```

[10]:

```
x1 = range(0, NUM_EPOCHS)
y1 = acc_list
plt.subplot(2, 1, 1)
plt.plot(x1, y1, 'o-')
plt.title('Train accuracy vs. epoches')
plt.ylabel('Train accuracy')
x2 = range(0, NUM_EPOCHS)
y2 = np.array(loss_list)/(len(train_dataloader)*4)
```

```
plt.subplot(2, 1, 2)
plt.plot(x2, y2, '.-')
plt.xlabel('Train loss vs. epoches')
plt.ylabel('Train loss per image')
plt.show()
#
```



[11]:

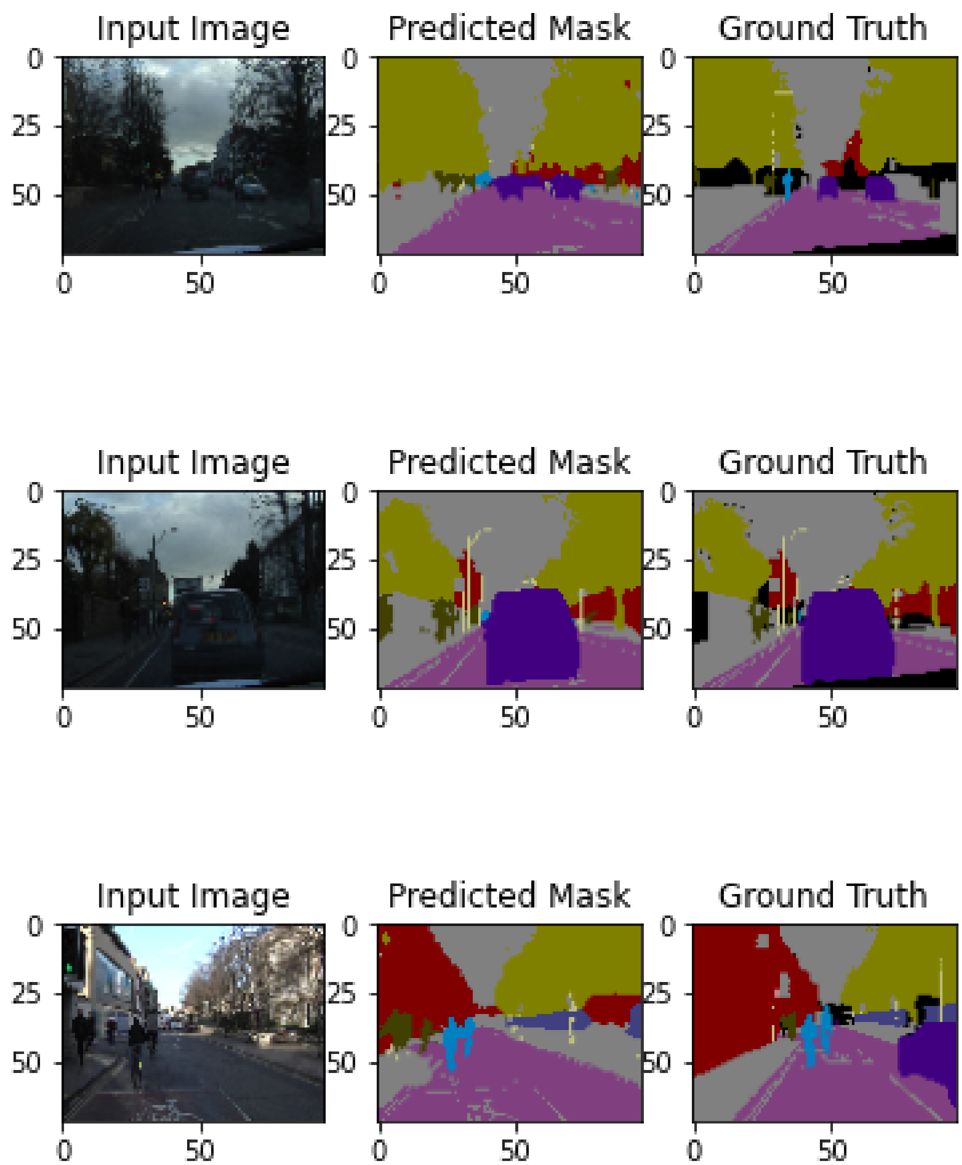
```
# test
SAVED_MODEL_PATH = "/content/drive/My Drive/segnet/model_best.pth"
OUTPUT_DIR = "/content/drive/My Drive/segnet/result_image"


# Parameters
NUM_INPUT_CHANNELS = 3
NUM_OUTPUT_CHANNELS = 13

MOMENTUM = 0.9
BATCH_SIZE = 4
CAMVID_CLASS_WEIGHTS = [0.58872014284134,
                 0.51052379608154,
                 2.6966278553009,
```

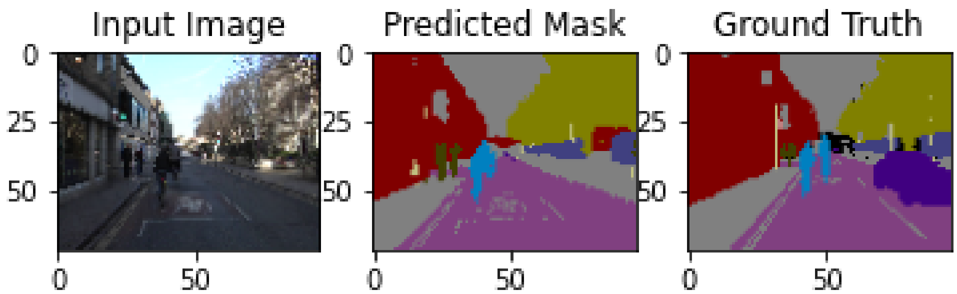
```
                  0.45021694898605,
                  1.1785038709641,
                  0.77028578519821,
                  2.4782588481903,
                  2.5273461341858,
                  1.0122526884079,
                  3.2375309467316,
                  4.1312313079834,
                  0.3,
                  0]
def validate():
    model.eval()
    acc_list_val = []
    loss_list_val = []
    for batch_idx,[img,label] in enumerate(val_dataloader):
        input_tensor = torch.autograd.Variable(img)
        target_tensor = torch.autograd.Variable(label)

        if CUDA:
            input_tensor = input_tensor.cuda()
            target_tensor = target_tensor.cuda()

        predicted_tensor, softmaxed_tensor = model(input_tensor)
        loss = criterion(predicted_tensor, target_tensor)
        loss_list_val.append(loss.float())

        _, prediction = torch.max(predicted_tensor,1)
        y_true = torch.flatten(target_tensor)
        y_prep = torch.flatten(prediction)
        intersection = torch.sum(y_true * y_prep)
        acc = (2. * intersection) / (torch.sum(y_true*y_true) + torch.
 ↪sum(y_prep*y_prep))
        acc = acc.detach().cpu().numpy()
        acc_list_val.append(acc)
        print("Processd #{} batch out of {}/{}".format(batch_idx+1, batch_idx+1,␣
 ↪8))

        if batch_idx == 8:
            print("Visualize the result for 3 images in last batch")
            for idx, predicted_mask in enumerate(softmaxed_tensor):

                target_mask = target_tensor[idx]
                input_image = input_tensor[idx]
                fig = plt.figure()

                a = fig.add_subplot(1,3,1)
                plt.imshow(input_image.cpu().permute(1,2,0))
```

```
                a.set_title('Input Image')

                a = fig.add_subplot(1,3,2)
                predicted_mx = predicted_mask.detach().cpu().numpy()
                predicted_mx = predicted_mx.argmax(axis=0)
                predicted_rgb = label_to_rgb(predicted_mx, color_encoding)
                plt.imshow(predicted_rgb)
                a.set_title('Predicted Mask')

                a = fig.add_subplot(1,3,3)
                target_mx = target_mask.detach().cpu().numpy()
                target_rgb = label_to_rgb(target_mx, color_encoding)
                plt.imshow(target_rgb)
                a.set_title('Ground Truth')

                fig.savefig(os.path.join(OUTPUT_DIR, "prediction_{}_{}.png".
 ↪format(batch_idx+1, idx)))
            break
    return loss_list_val, acc_list_val
if __name__ == "__main__":

    CUDA = torch.cuda.is_available()
    if CUDA:
        model = SegNet(input_channels=NUM_INPUT_CHANNELS,
                       output_channels=NUM_OUTPUT_CHANNELS).cuda()

        class_weights = torch.FloatTensor(CAMVID_CLASS_WEIGHTS).cuda()
        criterion = torch.nn.CrossEntropyLoss(weight=class_weights).cuda()
    else:
        model = SegNet(input_channels=NUM_INPUT_CHANNELS,
                       output_channels=NUM_OUTPUT_CHANNELS)

        class_weights = torch.FloatTensor(CAMVID_CLASS_WEIGHTS).cuda()
        criterion = torch.nn.CrossEntropyLoss(weight=class_weights)


    model.load_state_dict(torch.load(SAVED_MODEL_PATH))

    loss_list_val = []
    acc_list_val = []
    loss_list_val, acc_list_val = validate()
```

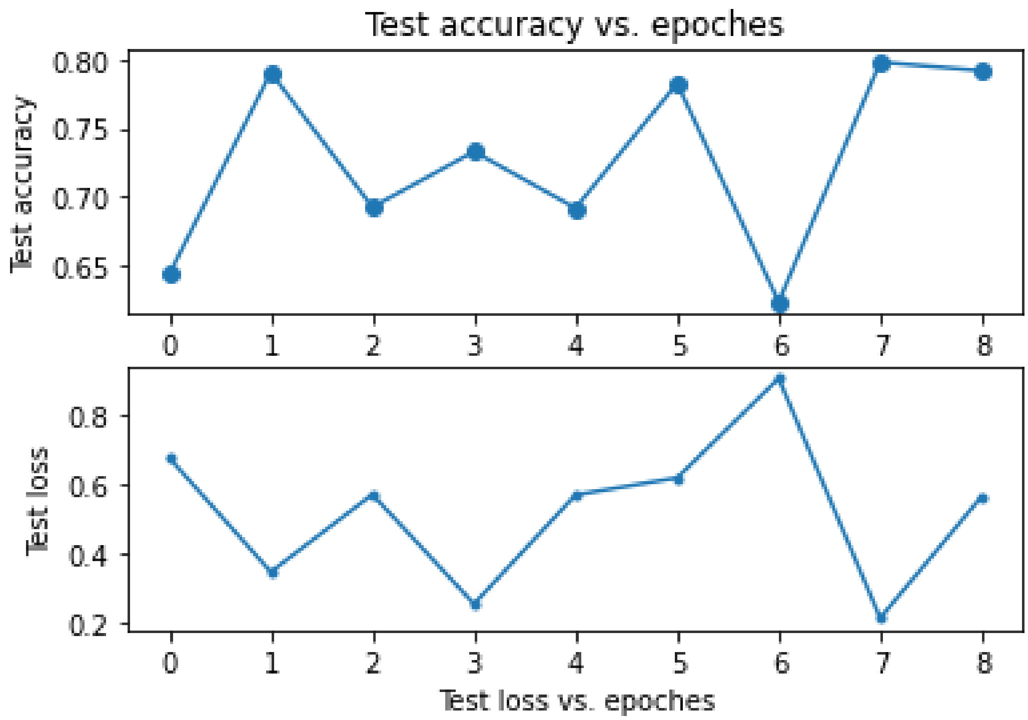
```
Processd #1 batch out of 1/8
Processd #2 batch out of 2/8
Processd #3 batch out of 3/8
Processd #4 batch out of 4/8
Processd #5 batch out of 5/8
```

```
Processd #6 batch out of 6/8
Processd #7 batch out of 7/8
Processd #8 batch out of 8/8
Processd #9 batch out of 9/8
Visualize the result for 3 images in last batch
```





[15]:
```
x1 = range(0, len(acc_list_val))
y1 = acc_list_val
plt.subplot(2, 1, 1)
plt.plot(x1, y1, 'o-')
plt.title('Test accuracy vs. epoches')
plt.ylabel('Test accuracy')
x2 = range(0, len(acc_list_val))
y2 = np.array(loss_list_val)/4


plt.subplot(2, 1, 2)
plt.plot(x2, y2, '.-')
plt.xlabel('Test loss vs. epoches')
plt.ylabel('Test loss')
plt.show()
#
mean_acc = np.sum(y1)/len(y1)
mean_loss = np.sum(y2)/len(y2)
print('Mean accurancy on validation set is {:2f}'.format(mean_acc))
print('Mean loss on validation set is {:2f}'.format(mean_loss))
```




```
Mean accurancy on validation set is 0.727296
Mean loss on validation set is 0.523363
```

0.7 Conclusion

From single image training and test result, we validate the usability of the segnet. And we can see the model overfitted on single image and result "perfect" predicted mask.

For whole dataset training, we conducted 160 epochs. The trend of the learning curve and accuracy curve are plausible (one converges to the apparent error one improves overall), but still because our selection of the learning rate and varying image distributions the curves are with some fluctuations.

For the validation part, we draw 8 mini batches from the validation set. The overall mean accuracy among all test batches turns to be 72.7%, comparing to the correspond value given in the paper is 61.9%, but that doesn't mean our result is well performed since the mean loss is not that low comparing to the overfitting single image loss. But by comparing the visualized results, we can manually check the real performance. The predicted mask and ground truth image are matched well, even same on a lot of detailed features. So my inference that the lower mean acc may be caused by some specific objects mis matching, e.g. in comparison images 3 and 4 the vehicle on the sideway are entirely not clustered at all.

There are still lots of improvements we can do on our reproduction project. Since we don't have official source code, we implemented lots of functions ourselves, some of them are time and space inefficient, which boosts the training time epically when the image resolution is high. That's also the reason we can't eventually train on the full-size data. Secondly, we didn't use advanced ML techniques in this project such as early stopping, varying learning rate and etc. But if those can be added we can see the pros and cons on using these methods.

For illumination invariance image processing implementing those two image processing methods, the RGB images can be successfully transformed into illumination-invariant grayscale images. It can be concluded from the processed images that the influence of varying illumination has been eliminated. For example, the intensity of a whole wall in the gray image will keep consistent even though the wall has half of it covered by the shadow. Since the original RGB images have three channels as the input of the neural network, the processed grayscale images need to be repeated for three times to act as the three channels of the input to fit the pre-trained VGG16 network. As for the code, the image processing function serves as part of the transformation function. Once a batch of images were chosen to be trained or tested, they would be transformed into illumination-invariant images. Unfortunately, the processed images could not be trained successfully because the prediction array always kept being 'nan' from the first epoch for some unknown reasons, which caused back propagation impossible.

0.8 Reference

1. G. D. Finlayson and S. D. Hordley, "Color constancy at a pixel," Journal of the Optical Society of America. A, Optics, Image Science, and Vision, vol. 18, no. 2, pp. 253–264, 2001.
2. W. Maddern, A. Stewart, C. McManus, B. Upcroft, W. Churchill, and P. Newman, "Illumination invariant imaging: Applications in robust vision-based localisation, mapping and classification for autonomous vehicles," in Proc. Int. Conf. on Robotics and Automation, vol. 2, 2014, p. 3.
3. J. A´ lvarez and A. Lopez, "Road detection based on illuminant invariance," IEEE Trans. on Intelligent Transportation Systems, vol. 12, no. 1, pp. 184–193, 2011.
4. V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39, no. 12, pp. 2481–2495, 2017.
5. K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," CoRR, vol. abs/1409.1556, 2014.
6. A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," in Advances in Neural Information Processing Systems 25, 2012.
7. https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c